

DUDLEY KNOX LIBRARY
NAVAL POSTGRADUATE SCHOOL
MONTEREY, CALIFORNIA 93943-5002

NAVAL POSTGRADUATE SCHOOL

Monterey, California



THESIS

A PARALLEL STRUCTURE
FOR ON LINE IDENTIFICATION
AND ADAPTIVE CONTROL

by

Yong Hong, Kim

March 1987

Thesis Advisor

Roberto Cristi

Approved for public release; distribution is unlimited.

T233100

REPORT DOCUMENTATION PAGE

1a REPORT SECURITY CLASSIFICATION UNCLASSIFIED			1b RESTRICTIVE MARKINGS		
2a SECURITY CLASSIFICATION AUTHORITY			3 DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution is unlimited		
2b DECLASSIFICATION/DOWNGRADING SCHEDULE			5 MONITORING ORGANIZATION REPORT NUMBER(S)		
4 PERFORMING ORGANIZATION REPORT NUMBER(S)			5 MONITORING ORGANIZATION REPORT NUMBER(S)		
6a NAME OF PERFORMING ORGANIZATION Naval Postgraduate School		6b OFFICE SYMBOL (If applicable) 62		7a NAME OF MONITORING ORGANIZATION Naval Postgraduate School	
8a ADDRESS (City, State, and ZIP Code) Monterey, California 93943-5000			7b ADDRESS (City, State, and ZIP Code) Monterey, California 93943-5000		
1a NAME OF FUNDING/SPONSORING ORGANIZATION		8b OFFICE SYMBOL (If applicable)		9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER	
1c ADDRESS (City, State, and ZIP Code)			10 SOURCE OF FUNDING NUMBERS		
			PROGRAM ELEMENT NO	PROJECT NO	TASK NO
			WORK UNIT ACCESSION NO		
1 TITLE (Include Security Classification) A PARALLEL STRUCTURE FOR ON LINE IDENTIFICATION AND ADAPTIVE CONTROL					
2 PERSONAL AUTHOR(S) KIM, Yong Hong					
3a TYPE OF REPORT Master's Thesis		13b TIME COVERED FROM TO		14 DATE OF REPORT (Year, Month, Day) 1987 March	
				15 PAGE COUNT 75	
5 SUPPLEMENTARY NOTATION					
7 COSATI CODES			18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)		
FIELD	GROUP	SUB-GROUP	Parallel structure for parameter estimation		
1 ABSTRACT (Continue on reverse if necessary and identify by block number)					
In this research study we investigate the possibility of employing parallel computing structures for adaptive identification and control. The identification algorithm is based on recursive least-squares with covariance resetting and block processing technique. It is shown how a parallel processing structure can be obtained by recursively applying Givens Rotation to the data matrix. Due to the regularity of its structure the approach presented is particularly attractive for VLSI implementation.					
11 DISTRIBUTION/AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS			21 ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED		
12a NAME OF RESPONSIBLE INDIVIDUAL Roberto Cristi			22b TELEPHONE (Include Area Code) 408-646-2223		22c OFFICE SYMBOL 62Cx

Approved for public release; distribution is unlimited.

A Parallel Structure
for On Line Identification
and Adaptive Control

by

Yong Hong, Kim
Major, Republic of Korean Air Force
B.S.E.E., Korean Air Force Academy, Seoul, 1978

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
March 1987

ABSTRACT

In this research study we investigate the possibility of employing parallel computing structures for adaptive identification and control. The identification algorithm is based on recursive least-squares with covariance resetting and block processing technique. It is shown how a parallel processing structure can be obtained by recursively applying Givens Rotation to the data matrix. Due to the regularity of its structure the approach presented is particularly attractive for VLSI implementation.

Thesis
4-21-45
5.1

TABLE OF CONTENTS

I.	INTRODUCTION	10
A.	BACKGROUND	10
B.	QR DECOMPOSITION	12
C.	SYSTOLIC ARRAY	13
II.	RECURSIVE LEAST-SQUARES ALGORITHM	14
A.	LINEAR MODELS FOR DISCRETE TIME SYSTEMS	14
B.	ON LINE PARAMETER ESTIMATION	15
C.	RECURSIVE LEAST-SQUARES ALGORITHM	18
D.	RECURSIVE LEAST-SQUARES WITH COVARIANCE RESETTING	20
III.	APPROACH TO PARALLEL ALGORITHM	23
A.	IMPLEMENTATION OF THE PARALLEL ALGORITHM	23
B.	SOLUTION OF THE LEAST-SQUARES PROBLEM USING QR DECOMPOSITION	24
C.	PARALLEL SOLUTION OF THE LEAST-SQUARES ALGORITHM	26
D.	SYSTOLIC ARRAY IMPLEMENTATION FOR PARALLEL STRUCTURES	29
	1. General Description of Systolic Arrays	29
	2. Triangular Systolic Array	31
	3. Linear Systolic Array	33
	4. Solving The Estimated Parameters	34
IV.	SIMULATION STUDY	39
A.	MODEL SET	39
	1. Noise Model	39
	2. Choice of Initial Value in The Systolic Array	40
B.	PARALLEL ALGORITHM SIMULATION	41
	1. Choice of Optimal Block Length	42

2.	Application to Systems with Varying Parameters	52
V.	PERFORMANCE COMPARISON AND CONCLUSION	62
A.	COMPUTATIONAL PROBLEM OF SERIAL RECURSIVE LEAST-SQUARES	62
B.	COMPUTATION OF PARALLEL STRUCTURE	63
	APPENDIX : COMPUTER PROGRAM OF PARALLEL ALGORITHM	66
	LIST OF REFERENCES	73
	INITIAL DISTRIBUTION LIST	74

LIST OF TABLES

1. REQUIRED BLOCK NO.	45
2. NUMBER OF ARITHMETIC OPERATION FOR UPDATING INPUT	62
3. NUMBER OF ARITHMETIC OPERATION IN PARALLEL STRUCTURE	63
4. COMPARISON OF SERIAL AND PARALLEL STRUCTURE (DIM. Φ = 4)	64

LIST OF FIGURES

2.1	Model Set	16
2.2	Adaptive Control	21
3.1	Two Step Computation for θ_{kN}	26
3.2	Systolic Array	29
3.3	Systolic Array Implementation for Recursive Least-Squares Algorithm	30
3.4	Computing Sequence of Triangular Array	31
3.5	Cell Definition for a Triangular Systolic for Performing Orthogonal Triangularization	32
3.6	Cells for Linear Systolic array	34
3.7	The Linearly connected Systolic Array	35
3.8	Overall Computation Structure	36
3.9	Block Processing	37
3.10	Operation of The Linear Systolic Array for $d = 4$	38
4.1	Noise Model -(a) Equation Error (b) Output Error	40
4.2	Example 4.1 - Parameter Estimation with $\sigma_o^2 = 1$	43
4.3	Example 4.1 - Parameter Estimation with $\sigma_o^2 = 100$	43
4.4	Example 4.2 - Parameter Estimation with $N = 5n$, W/O Noise	46
4.5	Example 4.2 - Parameter Estimation with $N = 15n$, W/O Noise	46
4.6	Example 4.2 - Parameter Estimation with $N = 20n$, W/O Noise	47
4.7	Example 4.2 - Parameter Estimation with $N = 5n$, $S/N = 10$	48
4.8	Example 4.2 - Parameter Estimation with $N = 15n$, $S/N = 10$	49
4.9	Example 4.2 - Parameter Estimation with $N = 20n$, $S/N = 10$	50
4.10	Example 4.3 - Tracking of Parameter with Time Varying, W/O Noise	53
4.11	Example 4.4 - Tracking of Parameter with Equation Error, $S/N = 10$	55
4.12	Example 4.4 - Tracking of Parameter with Equation Error, $S/N = 2$	55
4.13	Example 4.4 - Tracking of Parameter with Output Error, $S/N = 10$	56
4.14	Example 4.4 - Tracking of Parameter with Output Error, $S/N = 2$	57
4.15	Example 4.5 - Parameter Tracking with Time-Varying, $S/N = 10$ of Equation Error	60

4.16	Example 4.5 - Parameter Tracking with Time-Varying, $S/N = 10$ of Output Error	60
------	---	----

ACKNOWLEDGEMENT

I wish to express my appreciation to the Korean Air Force Authority for the opportunity to study in Naval Postgraduate School.

I would like to thank professor Roberto Cristi for being my thesis advisor. His tireless suggestions during the preparation of this thesis are very much appreciated. I got the most important ideas for this thesis from discussions with him.

I am also grateful to my wife, Hye Seon, for her endless patience and assistance.

I. INTRODUCTION

A. BACKGROUND

The techniques for on line identification have reached a great degree of sophistication in the recent years. Although many types of model structures could be used to describe the dynamical behavior of deterministic systems, the major emphasis is still on linear models. The whole problem is to try to determine which model best describes the dynamics under observation, in the neighborhood of the desired operating condition.

In some cases, insight of the model may be obtained from the laws of Physics, Chemistry, or similar. In most cases however, this might not be feasible due to the complexity of the physical factors involved. In these instances it may be possible to deduce the values of the parameters by observing the nature of the system's response under appropriate experimental conditions. We shall call this procedure parameter estimation.

The issue of parameter estimation plays an essential role in adaptive filtering, prediction, and control. The steps involved in a parameter estimation problem are the following:

1. Select an appropriate class of models.
2. Determine a "cost function", i.e., criteria by which the relative performance of different models within the class will be judged;
3. Choose the proper test signals which reflect all range of desired operating conditions. In particular for engineering applications we talk about signals "sufficiently rich" in frequency, which span the whole frequency range of interest.
4. Select an appropriate estimation algorithm according to on line or off line requirements. In the on line case, where the data are processed sequentially to yield an estimate in real time, the computational complexity is a fundamental factor in the choice of the estimation algorithm.
5. Make use of prior knowledge available on the system. It is always advantageous to incorporate as much prior knowledge as possible into the estimation algorithm, in terms of initial conditions and possible constraints in the estimated parameters.

For the particular case of estimating the parameters of linear models, we search within the class of models with a given fixed order for the one which minimizes a prediction error criterion. Test inputs are usually given in the form of white noise or sum of sinusoids within the frequency range of interest.

For what the choice of the estimation algorithm is concerned, extensive simulations show that techniques based on recursive least-squares present a much faster convergence rate than other algorithms, such as projection algorithm (Goodwin and Sin) [Ref. 1: p. 62]. The reason beyond this is the fact that the cost function associated with the former includes all past data (and so it has memory) while the latter is memoryless. The result is an improved convergence rate of the estimated parameters, an improved tracking for systems with time varying parameters, and better robustness in the presence of noise [Ref. 2: pp. 1-10] and unmodelled high frequency dynamics at the expenses of the need for more computing power.

The origins of the least-squares method can be traced back to Gauss [Ref. 3: p. 63]. In its recursive version it has been formulated independently by several authors. The original reference seems to be Plakett [Ref. 4: p. 149]. An early treatment of the least-squares method applied to dynamic system identification can be found in Aström [Ref. 5: pp. 123-130] and many related papers dealing with different aspects of the least-squares method can be found in the literature. A comprehensive treatment of adaptive techniques for identification is given in Ljung [Ref. 3: p. 17-20].

The major drawback of recursive least-squares identification is its cost in terms of computational complexity, which might make it unsuitable for real time identification of a large number of parameters. In fact it requires on line matrix manipulations, whose size grows with the dynamical complexity of the system to be estimated. The situation becomes even worse when on line identifications techniques are applied to multivariable systems, where the number of unknown parameters grow also with the number of inputs and outputs. In this respect the limitations due to serial computation is evident. From this list of implementational problems, the need for adequate computing capabilities is a crucial point for the implementation of on-line estimation techniques.

In this research we analyze an on line recursive least-squares identification algorithm which processes sequential "blocks" of data. In particular the parameter estimates are determined by the least-squares solution of a redundant number of linear equations obtained from the measured data. Existing techniques of solutions for this class of equations based on QR decomposition are particularly attractive for parallel processing applications. In the next section we introduce the concept of QR decomposition and its significance for parallel computation.

B. QR DECOMPOSITION

A numerically attractive method for the least-squares solution of linear equations is given by the QR decomposition of a matrix. To illustrate the concept consider the system of linear equations

$$A \theta = \underline{B} \quad (1.1)$$

in the unknown θ , with θ an n -dimensional vector and $\underline{B}$ m -dimensional, where $m > n$ (more equations than unknowns). The equal sign in (1.1) is intended to be in a least-squares sense. That is to say that given A , $\underline{B}$ the vector θ in (1.1) is the one which minimizes the square error $\|A\theta - \underline{B}\|$. By using the QR decomposition [Ref. 6: pp. 143-144] we can always decompose a given $m \times n$ matrix A as

$$A = Q\underline{R}$$

with Q an $m \times m$ orthogonal matrix, i.e., $QQ^T = I$, I being the identity matrix of appropriate dimensions, and $\underline{R}$ of the form

$$\underline{R} = \begin{vmatrix} R & \\ \text{---} & \\ 0 & \end{vmatrix}$$

with R $n \times n$ upper triangular. It can be shown that any solution of the equation (1.1) in the least squares sense is also solution of the system of n equations and n unknowns

$$R \theta = \underline{B}_1 \quad (1.2)$$

with $\underline{B}_1$ obtained from Q and $\underline{B}$. Also if the matrix A is full rank, the matrix R is invertible and the unique solution θ can be computed from (1.2).

The important points are the facts that equation (1.2) does not require explicit matrix inversion and can be solved by successive substitution, and a large number of techniques can be used to arrive to equation (1.2). In particular the Givens Rotation without square root [Ref. 7: pp. 215-218] to be presented in the subsequent chapters is attractive for systolic arrays implementation.

C. SYSTOLIC ARRAY

An efficient way of handling such procedure in QR decomposition is by using parallel processing structures, such as systolic arrays or wavefront arrays. The idea of a systolic array was developed by Kung and Leiserson. [Ref. 8: pp. 256-281] It consists of an array of individual processing cells that are arranged in the form of a regular structure. Each cell in the array is provided with local memory of its own and each cell is connected only to its nearest neighbors. The array is designed such that wavefronts of data are clocked throughout in a highly rhythmic fashion, much like the pumping action of a human heart; hence, the name "Systolic". These systolic arrays enjoy simple and regular communication paths, and almost all processors used in the networks are identical. As a result, special purpose hardware based on systolic arrays can be built inexpensively using VLSI technology. What is interesting in high-performance parallel structures is that it can be implemented directly as a dedicated hardware device.

In this research, the recursive algorithm, based on recursive least-squares with covariance resetting, is redesigned in order to make it suitable for implementation on a VLSI chip and also modify the systolic array in order to reduce computing time and complexity. The difference in adaptive control is that the estimator has to operate recursively on subsequent blocks of data, so that the estimated parameters converge asymptotically to the respective correct values. Asymptotic convergence is guaranteed by initializing each estimation with the parameter values from the previous one, and by persistency of excitation of the external inputs.

This research report is divided as follows; Chapter II discusses on line estimation and recursive least-squares, while Chapter III analyzes the parallel algorithm using QR decomposition and systolic array. In Chapter IV, shows the simulation results of parallel algorithm, comparison and conclusion is discussed in Chapter V.

II. RECURSIVE LEAST-SQUARES ALGORITHM

A. LINEAR MODELS FOR DISCRETE TIME SYSTEMS

Consider a dynamical system with single input signal $u(t)$ and single output signal $y(t)$. In a sampled data framework these are numerical sequences obtained from sampling continuous time signals at a frequency W_s . Without loss of generality assume the sampling interval T to be one time unit, and let the time index be $t = 1, 2, 3, \dots$. A linear discrete time model of the discrete time system is given by the difference equation

$$y(t) + a_1 y(t-1) + \dots + a_n y(t-n) = b_1 u(t-1) + \dots + b_m u(t-m) + v(t) \quad (2.1)$$

where $v(t)$ denotes a disturbance term to be specified. We shall use operator notation for conveniently writing difference equations. Thus let q^{-1} be the backward shift or time delay operator

$$q^{-1} x(t) = x(t-1) \quad (2.2)$$

Then (2.1) can be written as

$$A(q^{-1})y(t) = B(q^{-1})u(t) + v(t) \quad (2.3)$$

where $A(q^{-1})$ and $B(q^{-1})$ are polynomials in the delay operator:

$$A(q^{-1}) = 1 + a_1 q^{-1} + \dots + a_n q^{-n}$$

$$B(q^{-1}) = b_1 q^{-1} + b_2 q^{-2} + \dots + b_m q^{-m}$$

The model (2.1) or (2.3) describes the dynamic relations between the input and the output signals. In particular the polynomials A, B model the linear part, while the disturbance term v models disturbances (at the input and output), nonlinearities, and

anything which is not accounted by the linear model. An alternative way to represent the linear model is by defining the parameter vector.

$$\theta^T = (a_1, \dots, a_n, b_1, \dots, b_m)$$

the regression vector of lagged input-output data,

$$\phi^T(t) = [-y(t-1), \dots, -y(t-n), u(t-1), \dots, u(t-m)] \quad (2.4)$$

and write (2.3) as

$$y(t) = \theta^T \phi(t) + v(t) \quad (2.5)$$

Beyond the clear relation between (2.3) and (2.5), the significance of the regression model is two folded:

- a) it highlights the linear relationship between the observed signals $y(t)$, $\phi(t)$ and the parameters θ of the model, and
- b) if we assume $v(t)$ to be a sequence of n uncorrelated random variables (such as white noise) the predicted value of $y(t)$ given the measurements can be easily expressed as

$$\hat{y}(t) = \theta^T \phi(t) \quad (2.6)$$

In this research we are interested in the problem of estimating the parameter θ from measurements of the input,output sequences.

B. ON LINE PARAMETER ESTIMATION

It is possible to estimate the values of the model parameters θ by observing the nature of the system's response under appropriate experimental conditions. The idea for the recursive identification problem is to compare the output with that of an adjustable model, and update the parameters until the error between the outputs of plant and model is minimized. in some sense. The procedure is schematically described in Figure 2.1.

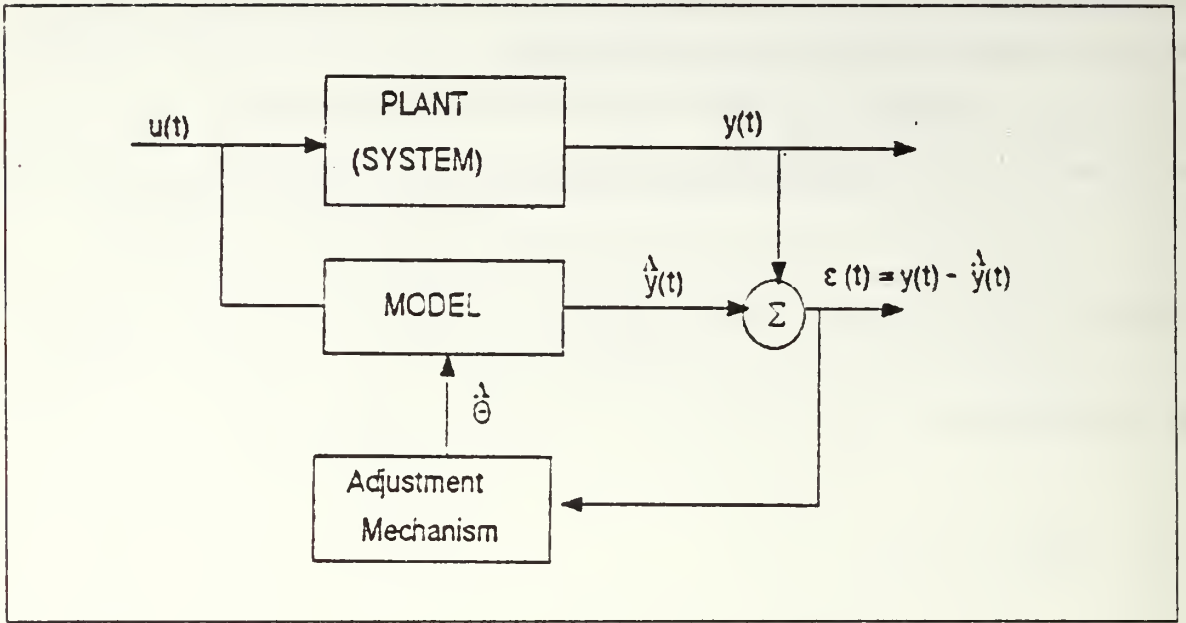


Figure 2.1 Model Set.

- Let us consider the difference equation model in (2.5). We desire to estimate the parameter vector θ from a set of measured data $y(t)$, $\varphi(t)$; $t = 1, 2, \dots, N$. In a least-squares framework we choose this estimate by best fitting the linear model to the available data. That is, we define a criterion function

$$V_N(\theta) = \frac{1}{N} \sum_{t=1}^N \alpha_t [y(t) - \theta^T \varphi(t)]^2 \quad (2.7)$$

to be minimized with respect to θ .

The inclusion of the coefficient α_t in the criterion (2.7) allows us to give different weights to different observations. In applications, most often α_t is either identically equal to 1 or expresses a forgetting factor of the form λ^t in order to give a higher weight to more recent data. [Ref. 3: p. 17]

If we denote by $\hat{y}(t|\theta) = \theta^T \varphi(t)$ as the prediction of $y(t)$, given the parameter vector θ , the criterion (2.7) can be seen as an attempt to choose a model that produces the best prediction for the output signal in the least-squares sense. The cost function V_N can be minimized analytically with respect to θ , to obtain the estimate

$$\hat{\theta}(N) = \left[\sum_{t=1}^N \alpha_t \varphi(t) \varphi^T(t) \right]^{-1} \sum_{t=1}^N \alpha_t \varphi(t) y(t) \quad (2.8)$$

which is defined, provided the inverse of the matrix involved exists. To obtain a recursive solution (2.8) denote

$$R(t) = \sum_{k=1}^t \alpha_k (\varphi(k) \varphi^T(k))$$

and, from (2.8), write

$$\sum_{k=1}^{t-1} \alpha_k \varphi(k) y(k) = R(t-1) \hat{\theta}(t-1).$$

From the definition of $R(t)$ it follows that

$$R(t-1) = R(t) - \alpha_t \varphi(t) \varphi^T(t)$$

hence standard algebraic manipulation lead to the recursion

$$\begin{aligned} \hat{\theta}(t) &= R^{-1}(t) \left[\sum_{k=1}^{t-1} \alpha_k \varphi(k) y(k) + \alpha_t \varphi(t) y(t) \right] \\ &= \hat{\theta}(t-1) + R^{-1}(t) \varphi(t) \alpha_t [y(t) - \hat{\theta}^T(t-1) \varphi(t)] \end{aligned} \quad (2.9)$$

The algorithm in (2.5) is not well suited for computation as it stands, since a matrix has to be inverted in each time step. It is more convenient to introduce the matrix

$$P(t) = R^{-1}(t)$$

and update $P(t)$ directly on the basis of the matrix inversion lemma [Ref. 3: p. 19] This leads to the recursion

$$P(t) = P(t-1) - \frac{P(t-1) \varphi(t) \varphi^T(t) P(t-1)}{1/\alpha_t + \varphi^T(t) P(t-1) \varphi(t)} \quad (2.10)$$

The advantage using of (2.10) is that the inversion of a square matrix of size equal to $\dim \theta$ is replaced by the scalar division. Thus the least-squares estimate $\theta(t)$ defined by (2.8) can be recursively calculated as

$$\begin{aligned}\hat{\theta}(t) &= \hat{\theta}(t-1) + \frac{P(t-1)\varphi(t)}{1/a_t + \varphi^T(t)P(t-1)\varphi(t)} [y(t) - \hat{\theta}^T(t-1)\varphi(t)] \\ P(t) &= P(t-1) - \frac{P(t-1)\varphi(t)\varphi^T(t)P(t-1)}{1/a_t + \varphi^T(t)P(t-1)\varphi(t)}\end{aligned}\quad (2.11)$$

The problem with this algorithm is that the matrix $R(t)$ might not be invertible and therefore $P(t)$ might not exist. This is the case when $t \leq \dim \varphi$, since in this case we can write

$$R(t) = [\varphi(1), \dots, \varphi(t)] \begin{bmatrix} \varphi^T(1) \\ \vdots \\ \varphi^T(t) \end{bmatrix} = \Phi_t \Phi_t^T$$

when Φ_t^T has more columns than rows (as in the case of $t \leq \dim \varphi$) we can always find a non zero vector $\underline{x}$ of appropriate dimension such that

$$\Phi_t^T \underline{x} = 0$$

which shows $R(t)$ to be a singular matrix whenever $t \leq \dim \varphi$. This problem of existence is addressed in the next section, where the recursive least-squares algorithm is modified to accommodate initial conditions and block processing techniques.

C. RECURSIVE LEAST-SQUARES ALGORITHM

The algorithm (2.11) can still be applied if we modify the cost function (2.7) as

$$V_N(\theta) = \frac{1}{2} \sum_{t=1}^N (y(t) - \varphi^T(t)\theta)^2 + \frac{1}{2} (\theta - \hat{\theta}(0))^T P(0)^{-1} (\theta - \hat{\theta}(0)) \quad (2.12)$$

where $P(0)$ and $\theta(0)$ are the initial conditions of P and θ respectively. Basically, the cost in (2.12) represents the sum of squares of errors $e(t) = y(t) - \varphi^T(t)\theta$, which is the difference between the actual observation $y(t)$ and the value predicted by the model with parameter vector θ . The second term on the right hand side of (2.12) has been included to account for the initial parameter estimate. Relevant observations for the recursive least-squares are given in the remainder of the section.

Observation 1: The recursive algorithm requires initial values for terms θ and P as $\theta(0)$ and $P(0)$ in order to start the recursion at $t=0$. The relative importance of the initial values on the recursive estimate decays with time, since the leftmost sum in (2.12) becomes preponderant with N encreasing. At the same time the entries of the matrix P become small with $t \rightarrow \infty$ for any initial conditions $P(0)$.

The initial value $P(0)$ of the matrix P appears in the cost function (2.12) as the weight given to the initial estimate $\theta(0)$, and it has to be positive definite. For convenience it is often chosen as $P(0) = C \cdot I$, where C is some positive constant. In particular $P(0)$ expresses the "confidence" we have on the initial conditions $\theta(0)$, and it is set to have large entries if we have poor confidence on the initial conditions $\theta(0)$. This can be easily seen in the cost function (2.12) where a very large $P(0)$ would give a very small weight to the term $\theta - \theta(0)$ associated with the initial conditions.

Observation 2: The least-squares algorithm generally has much faster convergence than other algorithms, such as the projection algorithm, and it can be used essentially unaltered with noisy signals [Ref. 1: P. 62]. We now discuss a number of variants of the least-squares algorithm. A problem of the recursive least-squares algorithms defined in (2.11) is that in general the matrix $P(t)$ tends to zero as $t \rightarrow \infty$. This can be seen from the definition of $P(t)$ in (2.11) and the matrix inversion lemma which yields

$$P(t)^{-1} = P(t-1)^{-1} + \varphi(t)\varphi^T(t)$$

in the scalar case $\varphi(t)\varphi^T(t)$ is a non-negative quantity, and therefore the sequence $P(t)^{-1}$ is nondecreasing and might tend to infinity (i.e., $P(t)$ might go to zero). The same result holds in the non-scalar case shown in (Goodwin and Sin) [Ref. 1: pp. 54-58]. The consequence of $P(t) \rightarrow 0$ is that the algorithm becomes less sensitive as $t \rightarrow \infty$. There are several ways to cope with this problem. We can either discount past data with a forgetting factor, or periodically reset the covariance matrix to nonzero constant values. The effect of these two modifications is the ability to apply the estimation algorithm to systems with time varying dynamics.

Although the algorithm presented and related analysis assumes a SISO plant for simplicity, they can be extended directly to the multivariable case (MIMO). Furthermore the plant is assumed to be causal, having unknown parameters and known order. The problem of order determination and validation involves criteria which go beyond the scope of this research. Therefore it will be always assumed that the order of the chosen model is a parameter given to the designer.

D. RECURSIVE LEAST-SQUARES WITH COVARIANCE RESETTING

The approach in this research is based on recursive least-squares with periodic covariance resetting, and block processing. By block processing approach we mean that the time set Z_+ is divided into segments I_k of fixed length N as

$$Z_+ = \bigcup_{k=1}^{\infty} I_k$$

where

$$I_k = \{ t \in Z_+ : (k-1)N \leq t < kN \}$$

and the parameter vector θ (the coefficients of A and B) in (2.3) are kept constant within each interval I_k .

The serial implementation call for the estimation $\theta(t)$ by the recursive least-squares shown in equation (2.11).

$$\begin{aligned} \hat{\theta}(t) &= \hat{\theta}(t-1) + \frac{P(t-1)\varphi(t)}{1 + \varphi^T(t)P(t-1)\varphi(t)} [y(t) - \hat{\theta}^T(t-1)\varphi(t)] \\ P(t) &= P(t-1) - \frac{P(t-1)\varphi(t)\varphi^T(t)P(t-1)}{1 + \varphi^T(t)P(t-1)\varphi(t)} \end{aligned} \quad (2.13)$$

and since the adaptive gains are updated at $t = kN$ only (once for every block), only the estimates θ_{kN} are considered.

There are several reasons which justify the block processing approach. For example one application of on line estimation is found in adaptive control.

As shown in Figure 2.2, in this class of problems, the output of the on line identifier is used to set the values of the parameters of a linear compensator. In an adaptive control context it has been shown that an external command $u(t)$ in (2.1) sufficiently rich in frequency, together with blocks I_k of sufficient length N provide a guarantee for a consistent estimation as

$$\hat{\theta}(t) \rightarrow \theta^* \quad (\theta^* \text{ representing actual parameters}).$$

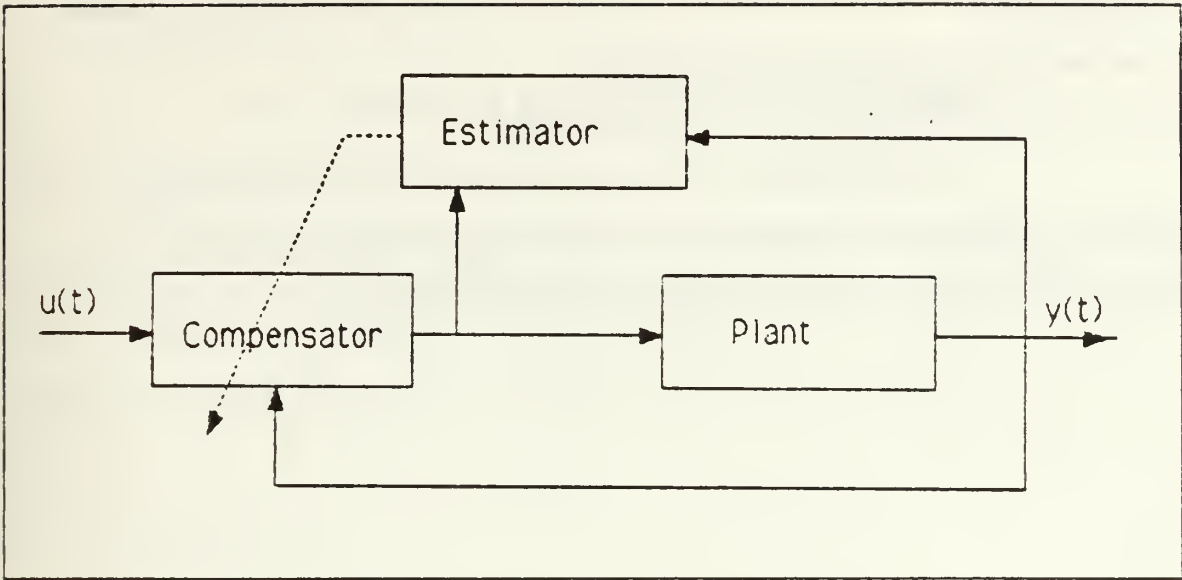


Figure 2.2 Adaptive Control.

As mentioned in the previous section the parameter estimation we consider is such that the covariance matrix $P_t \rightarrow 0$ and the estimator in (2.13) loses sensitivity to variation of θ^* as $t \rightarrow \infty$ [Ref. 1: p. 65]. Although the covariance matrix P_t can be reset at any time, for convenience of analysis we assume that it is reset at the beginning of each block as

$$P^{-1}_{kN-1} = \sigma_o^2 I \quad (2.14)$$

with I the identity matrix and σ_o an arbitrary positive constant. The result discussed above are formalized in the following theorem, proved in [Ref. 1: pp. 60-61].

Theorem

Let a SISO discrete time linear system be modeled by the difference equation (2.1) with $v(t) = 0$. Also let

1. $n = \text{degree } A(q^{-1}) = \text{degree } B(q^{-1})$.
2. $A(0) = 1, B(0) = 0$ (i.e. the system is causal).
3. let $\theta = [A, B]^T$, A, B being the coefficients of $A(q^{-1})$ and $B(q^{-1})$.
4. the coefficients of $A(q^{-1}), B(q^{-1})$ in (2.3) be held constant within the time intervals I_k , with $N = |I_k| \geq 10n$.

5. the external command sequence $u(t)$ have at least $4n$ different sinusoidal components.

6. the estimated sequence $\theta(t)$ be as in (2.13).

Then under these conditions, $\lim_{t \rightarrow \infty} \theta_t = \theta^*$ exponentially.

The extension to bounded disturbances $v(t) \neq 0$ can be shown under the condition that $|v(t)|$ is sufficiently small compared with the input and output signals.

III. APPROACH TO PARALLEL ALGORITHM

A. IMPLEMENTATION OF THE PARALLEL ALGORITHM

The real time identification algorithm with recursive least-squares, covariance resetting and block processing in equation (2.13) and (2.14) is redesigned for parallel implementation. By definition, using the minimization of quadratic cost function in (2.12), the estimate $\theta(t)$ at time t minimizes the quadratic cost function

$$V_{t,k}(\theta, \theta_{kN}) = \sum_{j=kN}^t |y(j) - \varphi(j)^T \theta|^2 + (\theta - \theta_{kN})^T P_{kN-1}^{-1} (\theta - \theta_{kN}) \quad (3.1)$$

with $t \in I_{k+1}$ (i.e., $kN \leq t < (k+1)N$) and $P_{kN-1}^{-1} = \sigma_o^2 I$. The fact that the covariance matrix P_t is reset to initial conditions at each $t=kN-1$, is equivalent to least-squares estimation in block I_{k+1} with initial conditions θ_{kN} . For $t = (k+1)N-1$ the cost function V in (3.1) can be written as

$$V_k(\theta) = W_k(\theta)^T W_k(\theta) \quad (3.2)$$

where

$$W_k(\theta)^T = [e_{kN}(\theta), \dots, e_{(k+1)N-1}(\theta), -\sigma_o(\theta - \theta_{kN})^T]$$

$$e_t(\theta) = y(t) - \varphi(t)^T \theta$$

By the above definitions we can write W_k as

$$W_k(\theta) = \left| \begin{array}{c} y((k+1)N-1) \\ \vdots \\ y(kN) \\ \hline \sigma_0 \theta_{kN} \end{array} \right| - \left| \begin{array}{c} \phi((k+1)N-1)^T \\ \vdots \\ \phi(kN)^T \\ \hline \sigma_0 I \end{array} \right| \theta \quad (3.3)$$

$$\Delta = \underline{g}_k - A_k \theta$$

with $\underline{g}_k \in R^{(d+N) \times 1}$, $\theta \in R^{d \times 1}$, $A_k \in R^{(d+N) \times d}$, where d is dimension of ϕ .

The algorithm we analyze computes θ_{kN} by minimizing the cost function in (3.1) based on a QR decomposition of the matrix A_k in (3.3). The estimated sequence θ_{kN} so obtained is therefore identical to the one obtained by recursive least-squares with covariance resetting in (2.13),(2.14) at $t = kN$.

B. SOLUTION OF THE LEAST-SQUARES PROBLEM USING QR DECOMPOSITION

A numerically attractive method of solving the least-squares error problem is the QR decomposition of a given matrix. By applying the QR decomposition method described in the introduction, at each time block k define Q_k and $\underline{R}_k$ as the QR decomposition factors of A_k as

$$Q_k A_k = \underline{R}_k, n \geq m \quad (3.4)$$

or equivalently

$$A_k = Q_k^T \underline{R}_k \quad (3.5)$$

with $Q_k \in R^{(d+N) \times d}$ an orthogonal matrix so that

$$Q_k^T = Q_k^{-1}$$

and $\underline{R}_k \in R^{(d+N) \times d}$ partitioned as

$$\underline{B}_k = \begin{bmatrix} R_k \\ O \end{bmatrix} \quad (3.6)$$

$R_k \in \mathbb{R}^{d \times d}$ upper triangular matrix, and O the null matrix. Applying the same transformation Q_k to $\underline{B}_k$ in (3.3) partition the result as

$$Q_k \underline{B}_k = \begin{bmatrix} \underline{B}_{k1} \\ \underline{B}_{k2} \end{bmatrix} \quad (3.7)$$

with $\underline{B}_{k1} \in \mathbb{R}^{d \times 1}$. By definition of V_k in (3.2) and orthogonality of Q_k we can write

$$V_k(\theta) = W_k^T(\theta) Q_k^T Q_k W_k(\theta)$$

and by in (3.3) - (3.7),

$$-Q_k W_k(\theta) = \begin{bmatrix} R_k \theta \\ O \end{bmatrix} - \begin{bmatrix} \underline{B}_{k1} \\ \underline{B}_{k2} \end{bmatrix}$$

where $\underline{B}_{k2}$ is independent of θ . Therefore

$$V_k(\theta) = \|R_k \theta - \underline{B}_{k1}\|^2 + \|\underline{B}_{k2}\|^2$$

In the above equation if the matrix R_k is nonsingular then the cost function V_k is minimized by

$$\theta_{(k+1)N} = R_k^{-1} \underline{B}_{k1} \quad (3.8)$$

Notice that equation (3.8) does not require any explicit matrix inversion since R_k is in upper triangular form. Therefore by the QR decomposition introduced above, we can compute $\theta_{(k+1)N}$ using two processors P1,P2 in cascade as in Figure 3.1.

In Figure 3.1, processor P1 computes R_k , $\underline{B}_{k1}$, while processor P2 computes θ from (3.8). In the next section we see how use parallel computational structure to compute R_k , $\underline{B}_{k1}$ and solve equation (3.8).

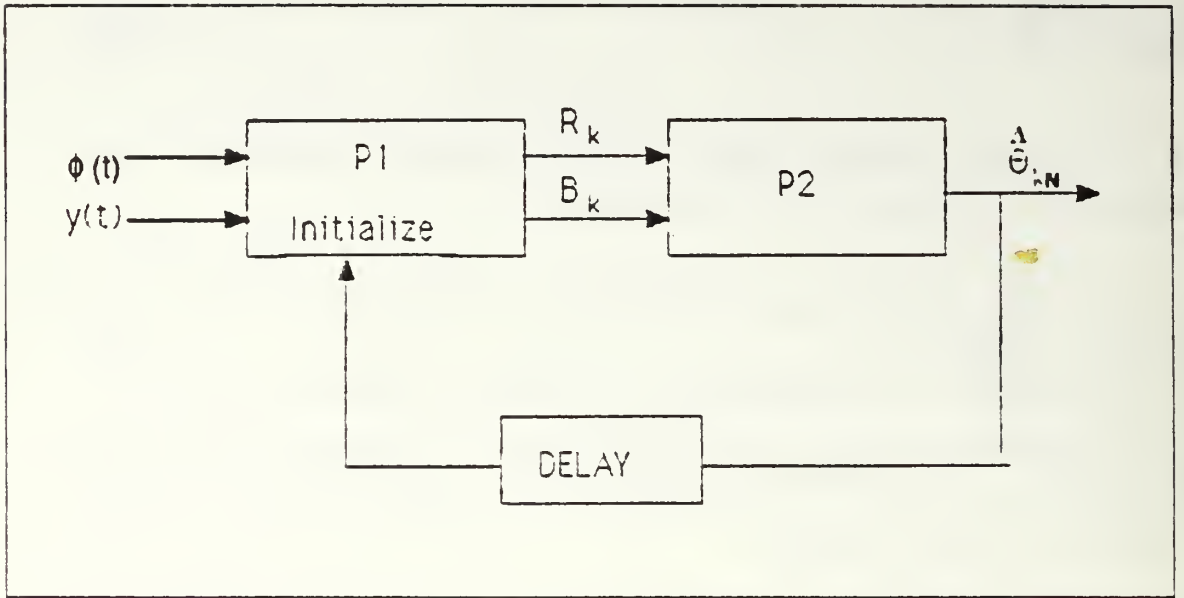


Figure 3.1 Two Step Computation for θ_{kN} .

C. PARALLEL SOLUTION OF THE LEAST-SQUARES ALGORITHM

This section is devoted to the solution of (3.8) using a parallel structure, and we introduce an algorithm suitable to the computation of the QR factorization of any given matrix of dimension $n \times m$ ($n > m$). Although several techniques exist in the literature, the Givens Rotation is preferred in parallel computing structures, since it successively manipulates two adjacent rows of the matrix at a time. For this purpose we define the Givens operator $Q(p,q)$, which performs a rotation of 2×1 dimensional subvectors of the matrix A in order to annihilate the element a_{qp} . In the applications to parallel processing, the rotation is determined based on adjacent elements of the first column, then it propagates to the successive right columns of the matrix. The Givens rotation provides matrix triangularization by affecting two rows at a time and the operations are limited to neighboring data in the matrix. Basis of the Givens rotation is the matrix

$$Q(p,q) = \begin{vmatrix} I_1 & 0 & 0 \\ 0 & r(p,q) & 0 \\ 0 & 0 & I_2 \end{vmatrix} \quad (3.9)$$

associated to each pair of indexes $p, q \in [1, d+1]$, with I_1 and I_2 identity matrices of dimensions $(q-2) \times (q-2)$ and $(d+1-q) \times (d+1-q)$ respectively, and $r \in \mathbb{R}^{2 \times 2}$ of the form

$$r(p, q) = \begin{vmatrix} c(p, q) & s(p, q) \\ -s(p, q) & c(p, q) \end{vmatrix} \quad (3.10)$$

Application of the transformation $Q(p, q)$ to any matrix A of appropriate dimensions, yields

$$Q(p, q)A = \begin{vmatrix} x & x & x \\ x & 1 & x \\ x & 0 & x \\ x & \underset{\substack{\uparrow \\ p}}{x} & x \end{vmatrix} \leftarrow q \quad (3.11)$$

provided the terms $c(p, q)$, $s(p, q)$ in (3.10) are such that

$$\begin{aligned} c(p, q) a_{q-1, p} + s(p, q) a_{qp} &= 1 \\ c(p, q) a_{qp} - s(p, q) a_{q-1, p} &= 0 \end{aligned} \quad (3.12)$$

which yields the unique solution of rotation parameters

$$\begin{aligned} c(p, q) &= \frac{a_{q-1, p}}{a_{q-1, p}^2 + a_{qp}^2} \\ s(p, q) &= \frac{a_{p, q}}{a_{q-1, p}^2 + a_{qp}^2} ; \end{aligned} \quad (3.13)$$

the notation x in (3.11) just indicates other terms of the matrix.

Observation: Notice that at each time block k , the matrix A_k as in (3.3) has full rank provided the parameter σ_0 is nonzero. Therefore the least-squares solution of (3.3) is always unique for each block k , and as a consequence the diagonal elements of R_k are always nonzero.

An example of application of the Givens Rotation in a three dimensional case is the following

$$\begin{vmatrix} r_{11} & r_{12} & r_{13} \\ 0 & r_{22} & r_{23} \\ 0 & 0 & r_{33} \\ 0 & 0 & 0 \end{vmatrix} = Q(3,4)Q(2,3)Q(1,2) \begin{vmatrix} \varphi_1 & \varphi_2 & \varphi_3 \\ a_{11} & a_{12} & a_{13} \\ 0 & a_{22} & a_{23} \\ 0 & 0 & a_{33} \end{vmatrix} \quad (3.14)$$

Each transformation Q in (3.14) is performed in two steps: At first, we compute c and s defined in (3.13) and then we perform the linear combination of rows q and $q-1$ of the A matrix in order to obtain the upper triangular matrix R_k in (3.14). By using this idea, we can implement the solution of (3.8) in a parallel structure. For each time block I_k define the initial matrices

$$R_k^{-1} = \sigma_o I \quad (3.15)$$

$$\underline{B}_{k1} = \sigma_o \theta_{kN}$$

θ_{kN} being the estimate from the previous block of data I_{k-1} . Then R_k in (3.8) can be determined as $R_k = R_k^{N-1}$ where R_k^j is recursively computed as

$$\begin{vmatrix} \frac{\varphi^T(kN+1)}{R_k^{j-T}} \end{vmatrix} = Q_k^j \begin{vmatrix} \frac{R_k^j}{0} \end{vmatrix}, j = 0, 1, \dots, N-1 \quad (3.16)$$

$$\begin{vmatrix} \frac{y(kN+j)}{\underline{B}_{k1}^{j-T}} \end{vmatrix} = Q_k^j \begin{vmatrix} \frac{\underline{B}_{k1}^j}{\underline{B}_{k2}^j} \end{vmatrix}$$

These operations are performed by the network of cell processors shown in Figure 3.2. The processors work simultaneously at the same clock pulse in each cell and their operations will be described by the transition and output function at the next section. In Figure 3.2, the regression vectors $\varphi(t)$ are at the top of the array, and the processors operate with the given data simultaneously. The details of the implementation are given in the next section.

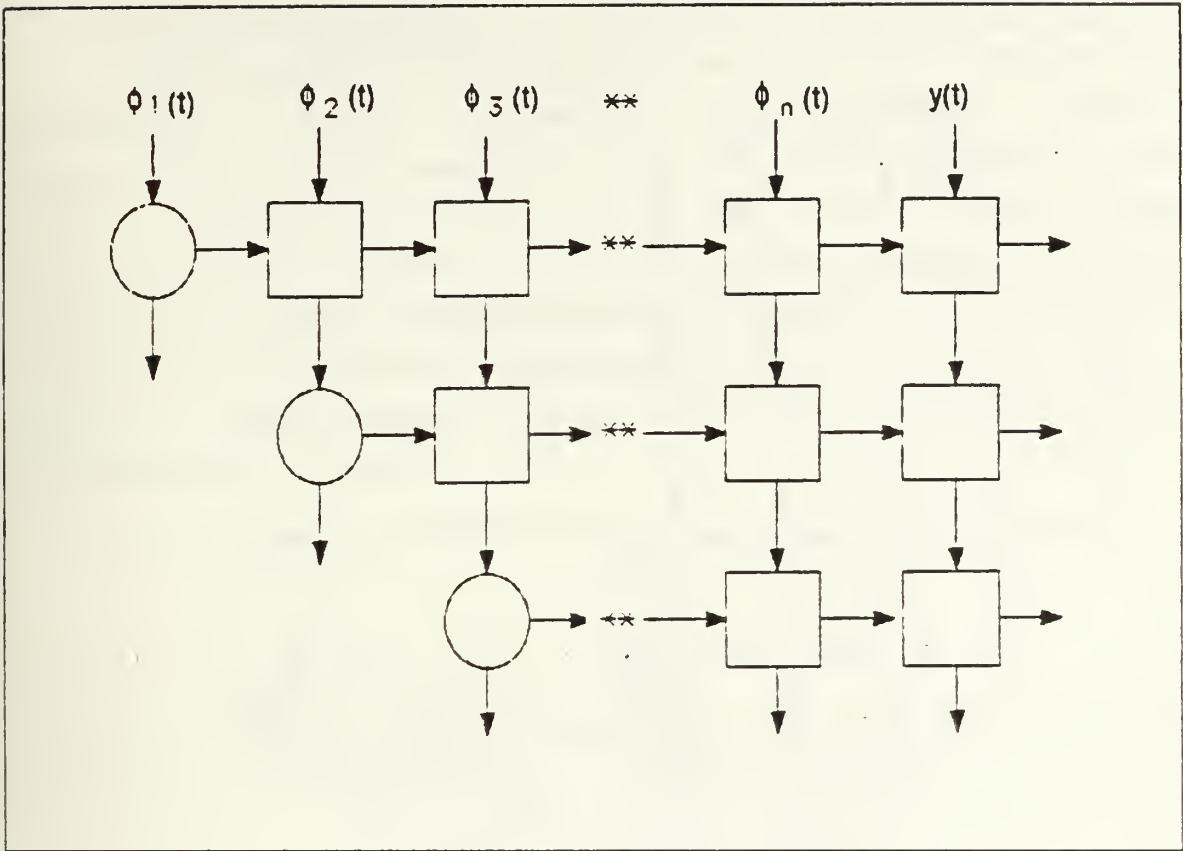


Figure 3.2 Systolic Array.

D. SYSTOLIC ARRAY IMPLEMENTATION FOR PARALLEL STRUCTURES

1. General Description of Systolic Arrays

In this section we show how the least-squares algorithm presented above can be implemented in a systolic array structure. In particular we aim at a computer structure which accepts as input the sequences of regression vectors $\phi(n)$ and signal $y(n)$ and outputs least-squares estimates of the parameter θ by block processing. The two processors described above (P1, P2. in Fig. 3.1) can be implemented as arrays of cells as shown in Figure 3.3, where the particular case of $d=4$ is presented. In the block processing we had mentioned in Chapter II, the processor P1(triangular section) operates during the given block time, while P2(linear section) operates only starting at the "reset" signal at the end of each block, all data in the triangular section flow to the linear section.

The entire systolic array is controlled by a single clock signal. Each array consists of two types of processing cells; internal cells (represented by squares) and

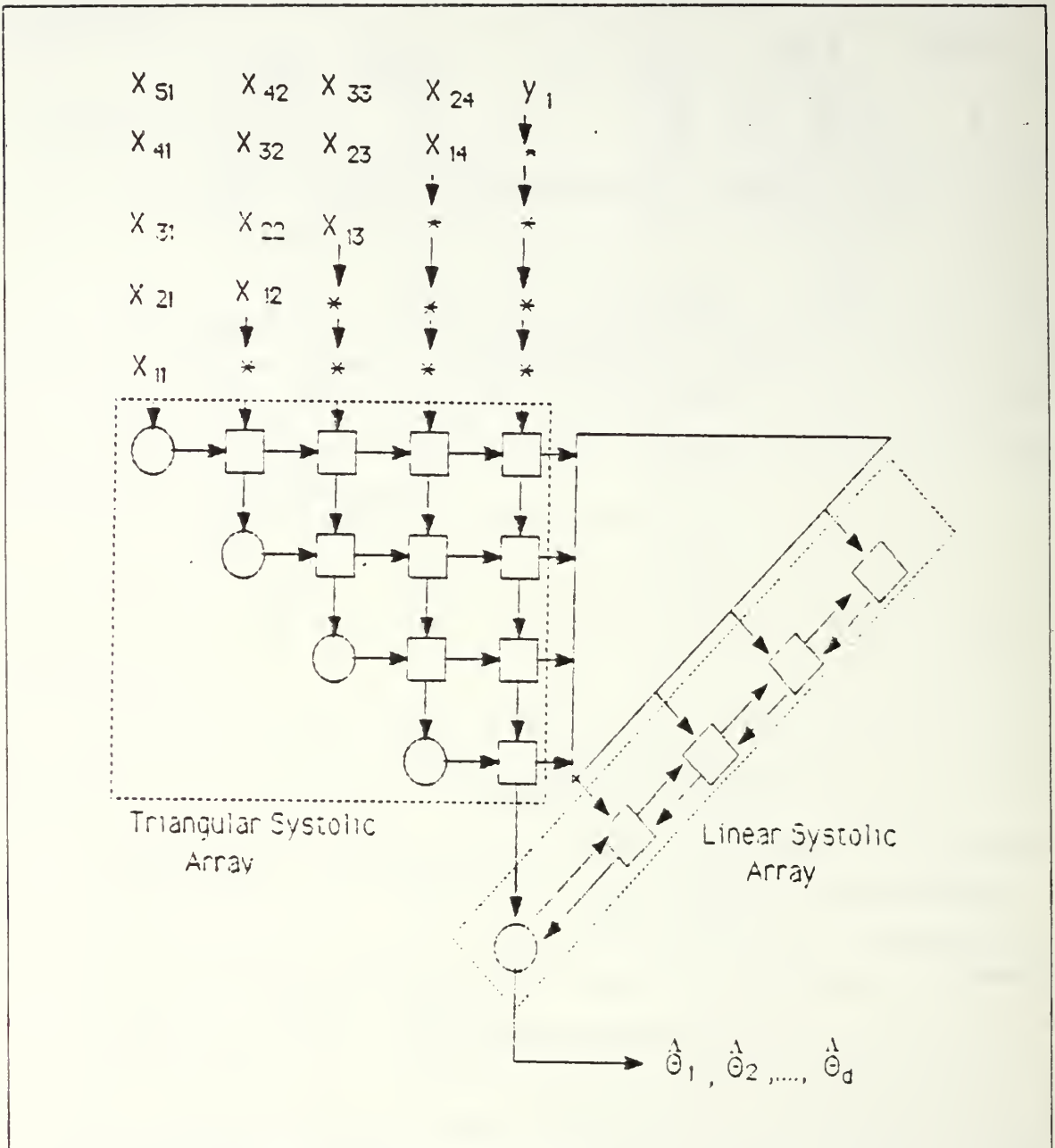


Figure 3.3 Systolic Array Implementation for Recursive Least-Squares Algorithm.

boundary cell (represented by circles). The specific arithmetic function of the cells will be defined later. The triangular systolic array section implements the Givens Rotation part of the recursive least-squares algorithm, whereas the linear systolic array section computes the least-squares weight vector at the end of the entire recursion without any matrix inversion.

2. Triangular Systolic Array

The triangular systolic array performs the Givens Rotation described above. In particular the boundary cells compute the rotation parameters c and s as in Equation (3.12), while the internal cells compute the linear combination of adjacent rows. The efficiency of the systolic array comes from the fact that each cell operates continuously on the data. In fact once a term in the matrix is updated and the results are passed to the cells on the right and below, new data can be immediately fed and computed. The mutual relation among the computing stages of different cells at the same time t is shown in Figure 3.3 where the label on each cell (i,j) indicates the order in which the data enter the array.

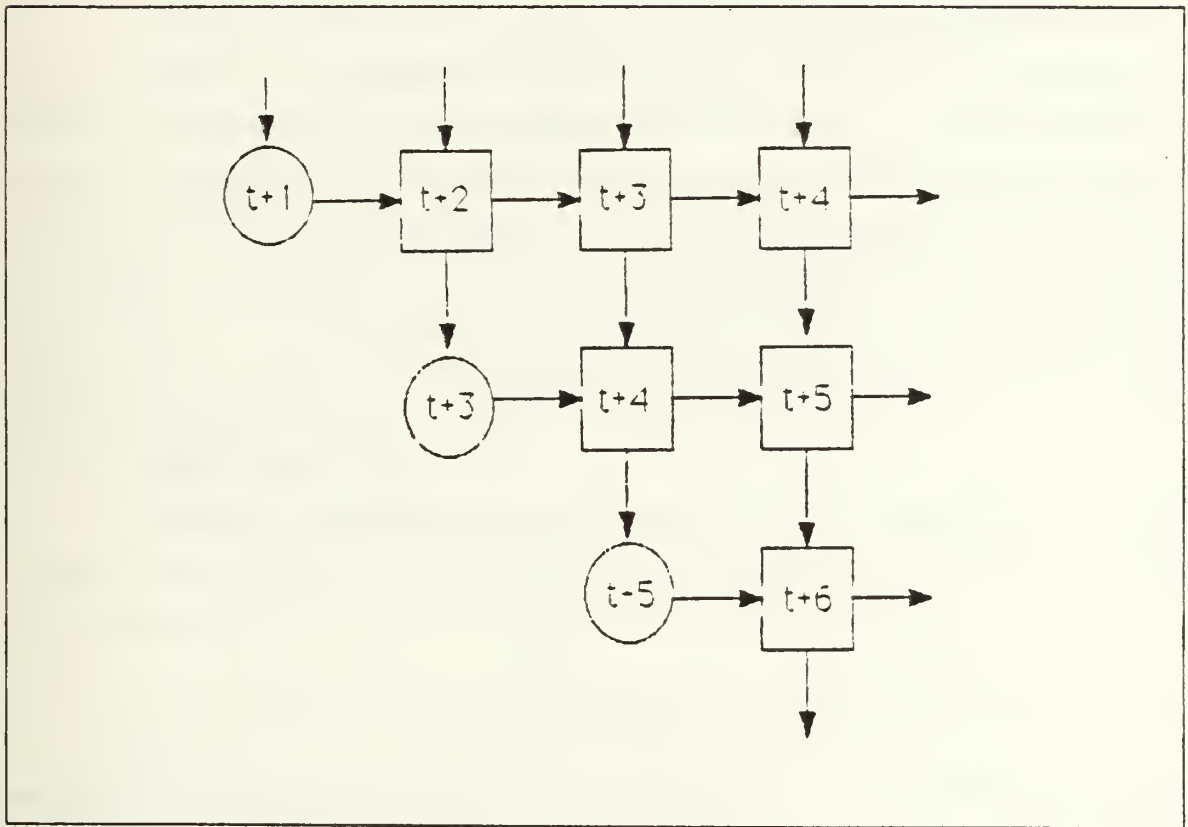
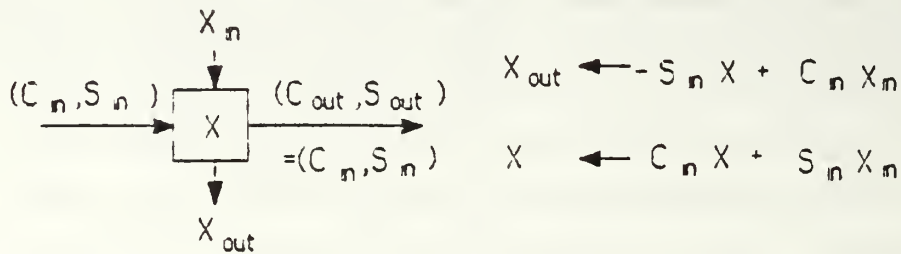


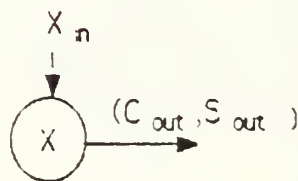
Figure 3.4 Computing Sequence of Triangular Array.

The operation of the triangular systolic array are summarized in Figure 3.5 which shows the boundary and internal cells. Basically, the internal cells perform only additions and multiplications, the boundary cells are considerably more complex in the sense that they compute three multiplications, one addition and one division at each cycle.

INTERNAL CELL :



BOUNDARY CELL :



if $X_n = 0$ then

begin

$C_{out} \leftarrow 1$

$S_{out} \leftarrow 0$

end

else

begin

$C_{out} \leftarrow 1 / 1 + X_n^2$

$S_{out} \leftarrow X_n / 1 + X_n^2$

$X \leftarrow 1$

end

Figure 3.5 Cell Definition for a Triangular Systolic for Performing Orthogonal Triangularization

Each cell of the triangular systolic array section stores a particular element of the upper triangular matrix $R(n)$ in (3.6) and it is initialized to zero for internal cells and σ_0 for the boundary cells as mentioned in (3.15). The function of each row of processing cells in the triangular systolic array section is to combine one row of the stored triangular matrix with a vector of data received from the above cells in such a way that the leading element of the received data vector is annihilated. The data

vector so obtained is then passed downward on to the next row of cells. The boundary cells in each row of the section computes the pertinent rotation parameters and then passes them to the right on the next clock cycle, as shown in Figure 3.5. The internal cells subsequently apply the same rotation parameter to all other elements in the received data vector.

Since a delay of one clock cycle per cell is incurred in passing the rotation parameters to the right along a row, it is necessary that the input data vectors enter the triangular systolic array in a skewed order, as illustrated in Figure 3.3. This arrangement of the input data ensures that each column vector $\phi(n)$ of the data matrix propagates through the array, it interacts with the previously stored triangular matrix $R(n-1)$ and undergoes the sequence of Givens Rotation $Q(n)$.

At the same time as the orthogonal triangularization process being performed by the triangular systolic array section, the column vector $\underline{B}(n)$ is computed by the rightmost column of internal cells. In effect, this computation is made by treating the desired response vector $Y(n)$ as an additional column that is appended to the data matrix $\phi(n)$ at its right end. When the entire orthogonal triangularization process is completed, each particular row of the upper triangular $d \times d$ matrix $R(k)$ and the associated $d \times 1$ vector $\underline{B}(k)$ along the corresponding wavefront is clocked out for subsequent processing by the linear systolic array section at the last clock time of the block processing period I_k .

3. Linear Systolic Array

The linear systolic array section consists of one boundary cell and $(d-1)$ internal cells that perform the arithmetic function defined in Figure 3.6 in accordance with equation (3.17).

$$\begin{aligned} Z_{d-i}(0) &= 0 \\ Z_{i-1}(n+1) &= Z_i(n) + r_{ij}(k) \theta_i(k) \\ \theta_i(k) &= \underline{B}_i(k) - Z_i(n) \end{aligned} \tag{3.17}$$

where i is from 1 to $d-1$, k is current NO. of block, n denotes the clock cycle in the linear section, $Z_i(n)$ are intermediate variables, and $r_{ij}(k)$ are elements of upper triangular matrix $R(k)$. $\underline{B}_i(k)$ are element of the vector $\underline{B}(k)$ and $\theta_i(k)$ are element of least-squares weight vector $\theta(k)$. This section of the processor computes $\theta(k)$ by using the method of backward substitution [Ref. 8: pp. 272-274].

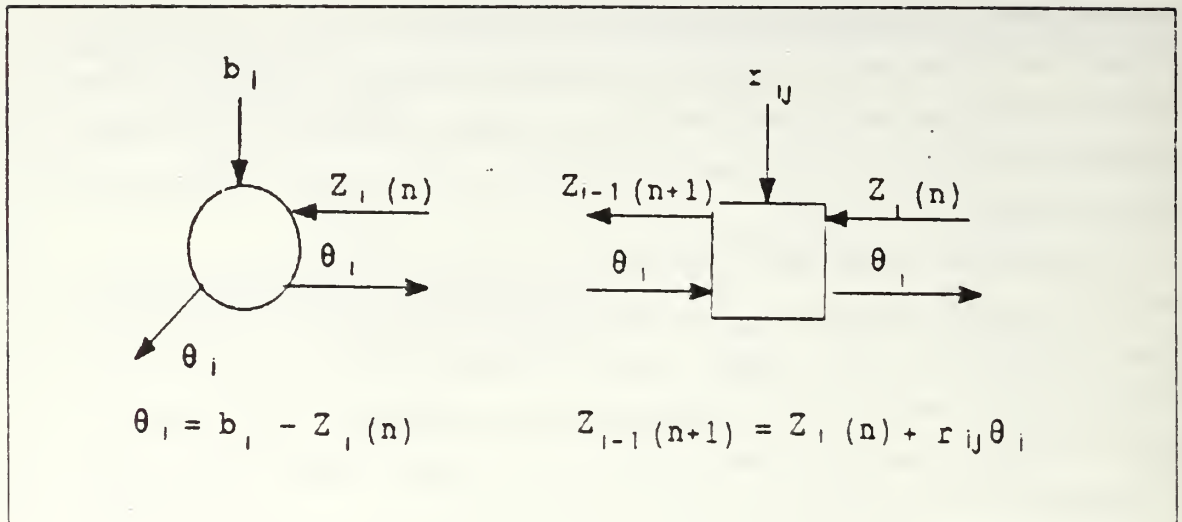


Figure 3.6 Cells for Linear Systolic array.

The boundary cell performs subtraction and division, whereas the internal cell perform additions and multiplications. The elements of the least-squares weight vector appear at the output of the boundary cell at different clock cycles, with $\underline{g}_i(k)$ leaving this cell first, followed by $\underline{g}_{i-1}(k)$, and so on right up $\underline{g}_1(k)$. In effect, the element of the weight vector $\theta_i(k)$ are read out backward, i.e., $i = d-1, \dots, 2, 1$.

4. Solving The Estimated Parameters

Once the triangular matrix $R(k)$ and the corresponding vector $\underline{g}(k)$ are computed, next task is to transfer the data to the second processor(the linear array) and solve for the estimated parameters. This operation is performed at the end of each time block($t = kN$). In order to shift the entries of the matrix $R(k)$ and vector $\underline{g}(k)$ of the triangular array it is easy to see from the cell processors in Figure 3.5, that values $c=0, s=-1$ of the rotation parameters cause the data in the cells to shift to the cell below. This operation can be successively propagated to all cells on the right. This operation is triggered by the "reset" command at the end of each time block. The reset command is given as an impulse to the boundary cells in the triangular systolic array. At the next of the reset command, the rotation parameters in the boundary cells are generated the values $c=1, s=0$ and pass them to rightward in order to shift down all stored datas in the triangular section without any computation in the internal cells. Let us consider the case where the dimension of $\phi(k)$ is 4($d=4$). After the "reset" command, the entries of 4×4 matrix $R(k)$ and 4×1 vector $\underline{g}(k)$ are shifted down to

the linear array with time sequence as shown in Figure 3.4. All cells in the linear array are initialized to zero. The variable $Z_i(n)$ are forced leftward through the processor with accumulation inner product terms in the rest of the processor as it moves to the left in Equation (3.17) while r_{ij} and b_i are lowered as indicated in Figure 3.7. The left end processor is different in that it computes $\theta_i(k)$ by function in equation 3.17. At any time b_i reaches the boundary cell, it is combined with $Z_i(n)$ and the processor computes the weight vector $\theta(k)$.

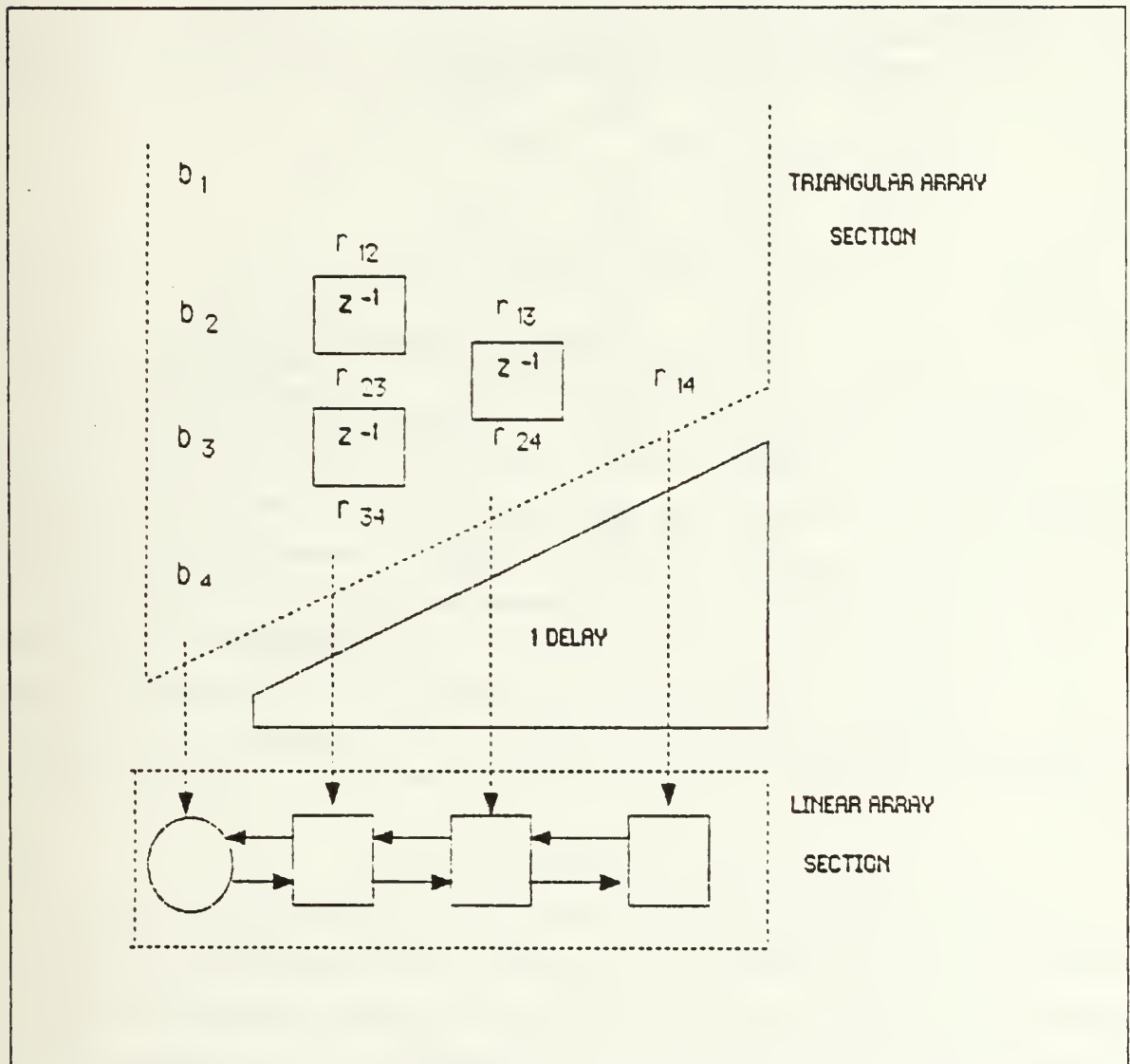


Figure 3.7 The Linearly connected Systolic Array.

We generate the element $\theta_i(k)$, $\theta_{(i-1)}(k)$ and $\theta_1(k)$ of the least-squares weight vector $\theta(k)$, in that order, at the output of the boundary cell, and transferred the matrix $B(k)$ as a part of the initial condition of the next block processing I_{k+1} as shown in Figure 3.8. [Ref. 10: pp. 514-515]

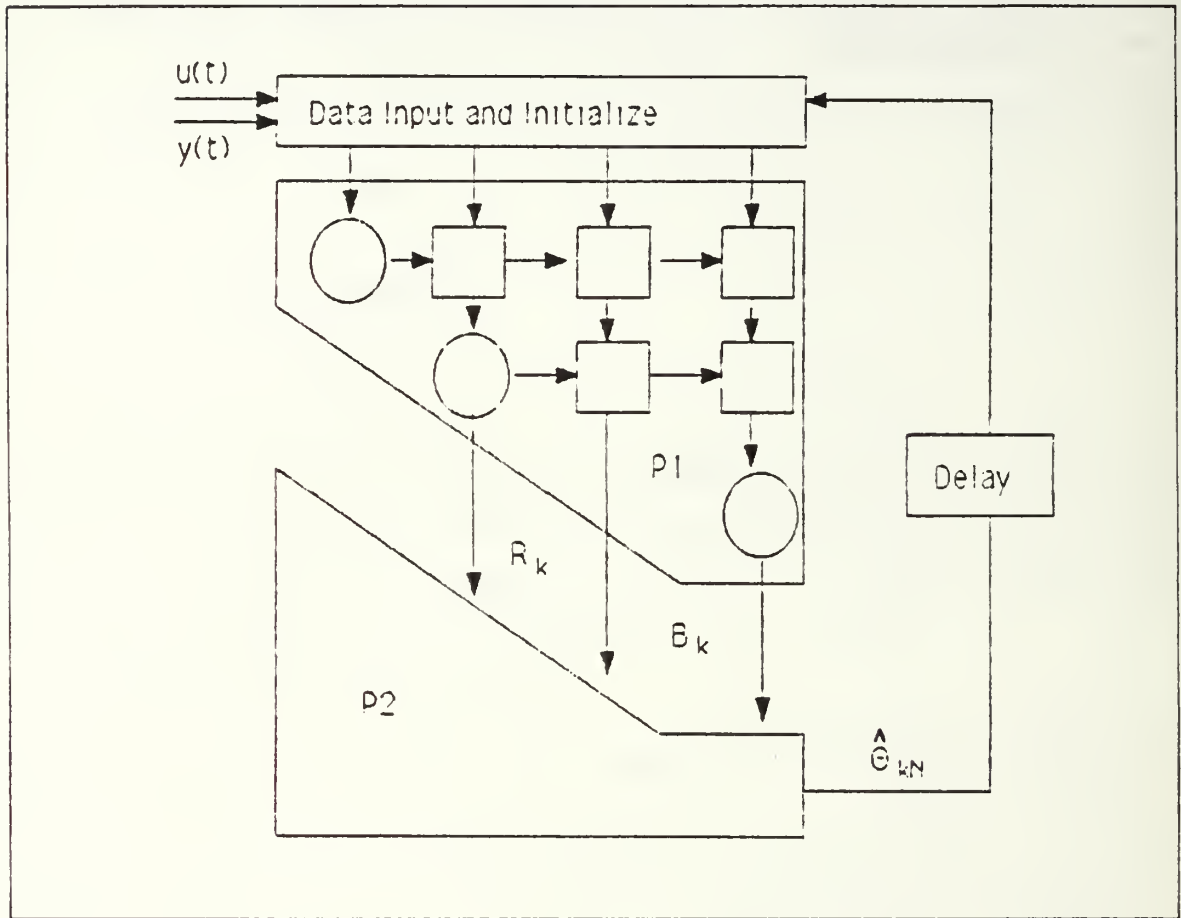


Figure 3.8 Overall Computation Structure.

In the case of SISO, the triangular systolic array operates during the interval I_k . Another time period is required for computation of the parameter $\theta_i(k)$ in the linear systolic array section. It turns out that the exact additional clock times for the linear systolic section (represented by I_1) is proportional to the dimension of $\varphi(n)$ ($I_1 = 2d$) and can be expressed as a $O(d)$ which is showed in Figure 3.9. In fact the solution of the triangular system requires a number of d substitutions.

To summarize, the total clock time required for computation of $\theta(k)$ is $N + 2d$. The operation involved at the individual clock cycles of the linear array section are described in Figure 3.10 for the case of $d = 4$.

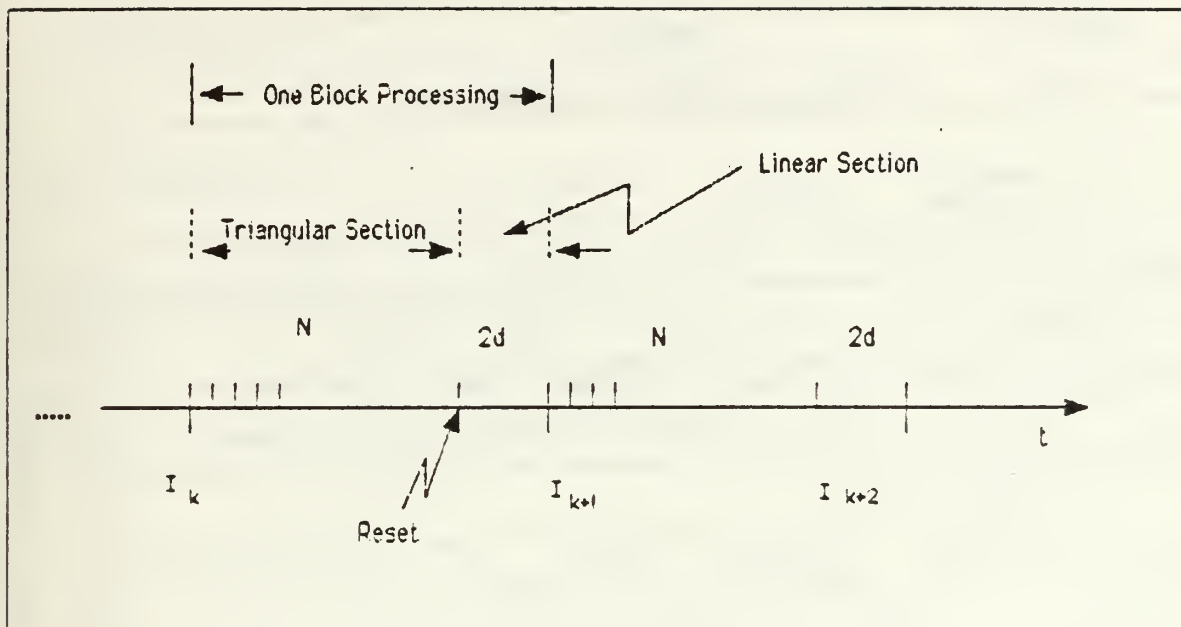


Figure 3.9 Block Processing.

From the above description we see that advantages and disadvantages are associated to this parallel estimation scheme. A clear advantage is the parallelism in computation, as opposed to the serial nature of recursive least-squares, which makes it attractive when a large number of parameters has to be estimated. A disadvantage is that this technique is effective only in conjunction with block processing. This is due to the fact that we always need the time to solve for the linear system in the linear section. Therefore the parameter estimates are updated only at the end of the block time, contrary to the serial case when updated estimates of the parameters are available at any time.

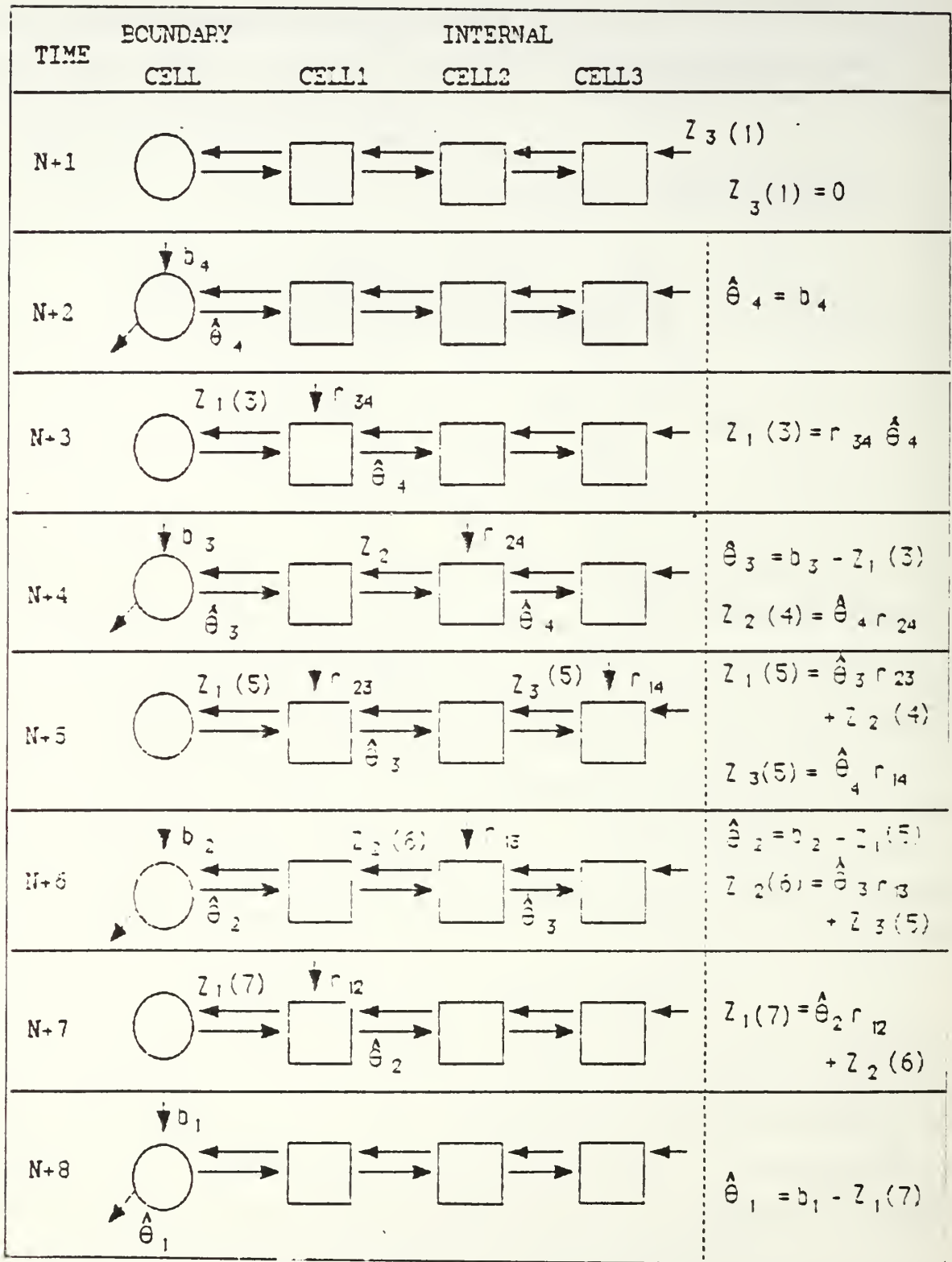


Figure 3.10 Operation of The Linear Systolic Array for $d = 4$.

IV. SIMULATION STUDY

A. MODEL SET

The linear model in general can be expressed by either the difference equation relating the input and output sequences, or in a regression form as in the previous Chapters. In the examples below we consider the problem of estimating the parameters of a third order model using the systolic technique described in the previous Chapter. Consider the class of models with difference equation

$$y(t) + a_1 y(t-1) + a_2 y(t-2) + a_3 y(t-3) = b_1 u(t-1) + v(t) \quad (4.1)$$

where $u(t)$, $y(t)$ are the input and output sequences and $v(t)$ is an added disturbance. An alternative way to express (4.1) is obtained by defining the vectors

$$\theta^T = (a_1, a_2, a_3, b_1) \quad (4.2)$$

$$\phi^T(t) = [-y(t-1), -y(t-2), -y(t-3), u(t-1)]$$

which leads to the regression model

$$y(t) = \theta^T \phi(t) + v(t) \quad (4.3)$$

This model describes the observed variable $y(t)$ as linear combination of the components of the observed vector $\phi(t)$ plus noise. In the parameter estimation problem the values in θ are assumed to be unknown.

1. Noise Model

The most realistic model has to include noise in system and measurement. In general, $v(t)$ in equation (4.1) is a sequence of independent random variables with zero mean values, for an ideal "white noise" source. When the noise is "colored" (i.e., its samples are correlated) we need to encrease the order of the difference equation to

include the dynamics of the noise model. In this case a higher model order is required which incorporate also noise dynamics [Ref. 3: p. 265]. From a physical point of view it is common to work with disturbances that are "measurement errors" or "output error" expressed in equation (4.4)

$$y(t) = \frac{B(q^{-1})}{A(q^{-1})}u(t) + v(t) \quad (4.4)$$

the difference between this model and equation 4.1 is also illustrated in Figure 4.1.

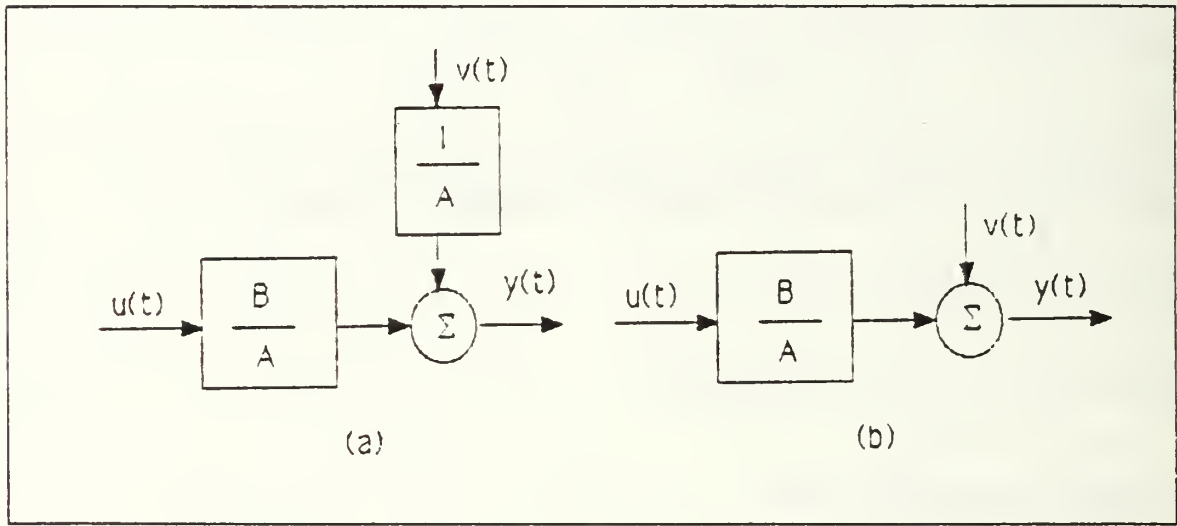


Figure 4.1 Noise Model -(a) Equation Error (b) Output Error.

Identification methods that use the model in equation (4.4) are often known as output error methods or model reference identification methods. In this research, we simulate two types of model using parallel identification algorithm.

2. Choice of Initial Value in The Systolic Array

For a recursive algorithm, we need to initialize the systolic array with an initial parameter estimate $\theta(0)$ and initial covariance matrix ($\sigma_0 I$ in equation 2.14). The choice of these initial values will be discussed in this section. In equation 2.14, σ_0 is related to the confidence we have on the initial condition $\theta(0)$. If some prior information about $\theta(0)$ is available it should of course be used for determining suitable values of $\theta(0)$. In particular as we have mentioned in Chapter III at the beginning of

the new block I_{k+1} , we use the estimate obtained at the previous time block. If no prior information is available, the most common choice is to take

$$\theta(0) = 0, P^{-1}(0) = \sigma_0^{-2}I \quad (4.5)$$

where σ_0^{-2} is a large number for fast convergence. In this research we use different values of $\sigma_0(0)$ for comparison of the results.

B. PARALLEL ALGORITHM SIMULATION

The program which is included in this research report in the Appendix is used to investigate the parallel algorithm on the basis of the systolic array presented in the previous chapters. Consider a plant with discrete time transfer function

$$H(z) = \frac{b_1 z^2}{z^3 + a_1 z^2 + a_2 z + a_3} \quad (4.6)$$

The plant has two zeros at $z = 0$ and three stable poles in the z -plane. The difference equation associated with the transfer function (4.6) can be expressed as

$$y(t) + a_1 y(t-1) + a_2 y(t-2) + a_3 y(t-3) = b_1 u(t-1) \quad (4.7)$$

For identification purpose the input is "sufficient rich" in frequency in order to excite all modes of the system. This translates in the requirement that the input $u(t)$ must contain a sufficient number of sinusoids. In particular $u(t)$ should contain at least $4n$ different sinusoidal components as mentioned in Chapter II. Writing (4.7) in regression form we obtain

$$y(t) = \phi^T \theta + v(t) \quad (4.8)$$

where

$$\phi^T(t) = [y(t-1), y(t-2), y(t-3), u(t-1)]$$

and

$$\theta = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ b_1 \end{bmatrix}$$

θ being the vector of the actual parameters of the plant.

Example 4.1)

Consider the discrete time system with transfer function,

$$H(z) = \frac{1.3 z^2}{(z - 0.5)^5}$$

linear difference equation

$$y(t) - 1.5y(t-1) + 0.75y(t-2) - 0.125y(t-3) = 1.3u(t-1) \quad (4.9)$$

and input sequence

$$u(t-1) = \cos(n\pi/10) + \cos(n\pi/5) + \cos(n\pi/2) + \cos(3n\pi/5).$$

We applied the parallel identification algorithm using $\sigma_0 = 1$, and 100, no disturbance present and $N = 10n$ as the block length for the triangular systolic array section. The corresponding plots are given in Figures 4.2 and 4.3. We can see that in ideal conditions all estimated parameters converge to their correct values. The estimated weight vector plotted versus time for number of block (represented by NB).

1. Choice of Optimal Block Length

In the previous chapters, we set the time block length to $N \geq 10n$. The reason of this has been highlighted in Chapter I, and it gives a sufficient condition for global stability in adaptive control. In the simple case of parameter estimation we do not have to maintain this restriction. An important factor is to achieve fast convergence, whenever possible and to find an optimal block length for fast convergence. The influence of the block length on the convergence rate will now be illustrated by means of simulation in example 4.2.

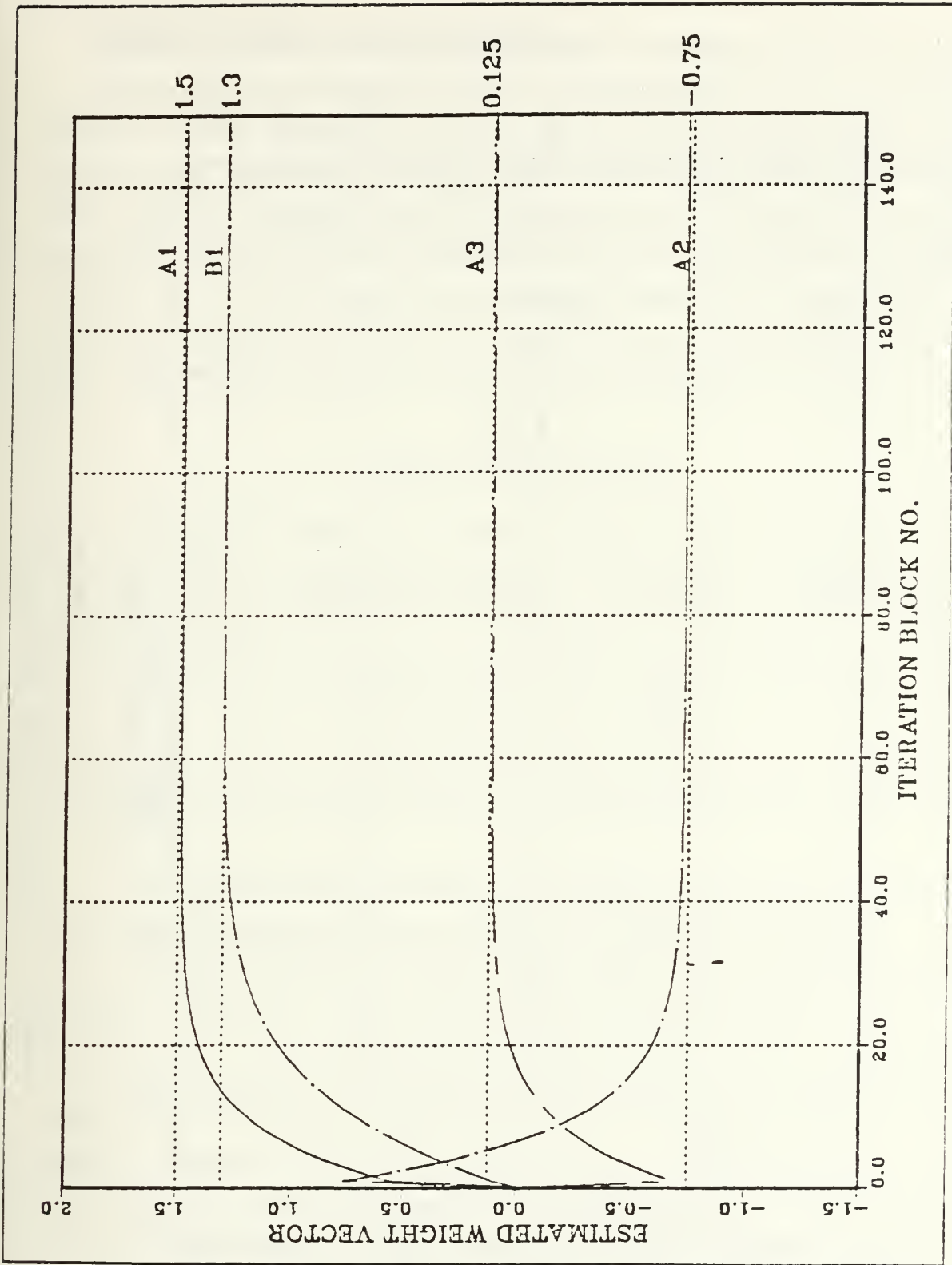


Figure 4.2 Example 4.1 - Parameter Estimation with $\sigma_0^2 = 1$.

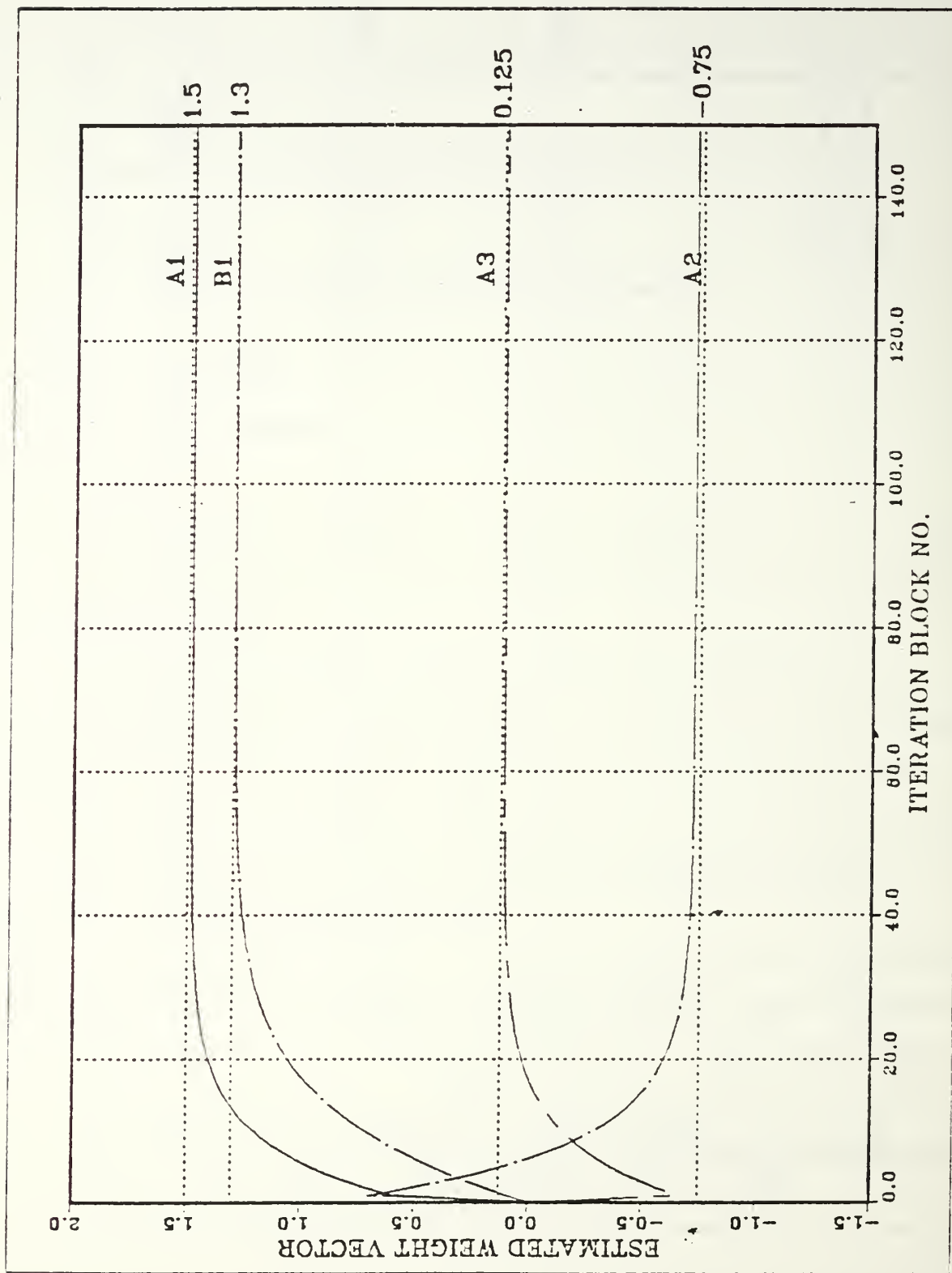


Figure 4.3 Example 4.1 - Parameter Estimation with $\sigma_0^2 = 100$.

Example 4.2)(choice of optimal block length)

For the illustration of optimal length, we have used the same system shown in equation (4.9) with different block lengths. It has been simulated with 3 different block lengths without noise and with noise condition from equation error ($S/N = 10$) in order to test the convergence rates in the different cases. So, we have used the same amount of clock cycles for the comparison of each results using different block lengths. It means that the scale of the X-axis has exactly the same amount of clock cycles. The numerical computation for the required number of blocks are summarized in Table 1 in order to expressed the number of time blocks in each simulation.

TABLE 1
REQUIRED BLOCK NO.

Block Length N	Entire Block Length Per One Block ($N + 2d$)	Total Clock Cycles	Required Block NO.
5n	$15 + 8 = 23$	5704	248
10n	$30 + 8 = 38$	5700	150
15n	$45 + 8 = 53$	5724	108
20n	$60 + 8 = 68$	5712	84

The results are shown in Figures 4.4 through 4.6 without noise condition, and the corresponding outputs with noise condition are in Figures 4.7 through 4.9.

It turns out that the estimated weight vector shows the fastest convergence rate when used with $N = 15n$. Even though the estimated parameter a_1 converges very fast at $N = 5n$, other estimated parameters exhibit slow convergence rate compare to the case of $N = 15n$, regardless the noise condition. Therefore the optimal choice of block length must be the case when $N = 15n$, and there is one more interesting comment in these results. We realized that within certain limits, the longer the block length, the faster the convergence rate, but when we extended the length $N = 20n$, the convergence rate of that result is similar to the output of $N = 10n$, and for the more accurate comparison we can look at the Figure 4.3 ($N=10n$) and 4.7 ($N=20n$). As it were, the conclusion with this example is that the convergence rate to

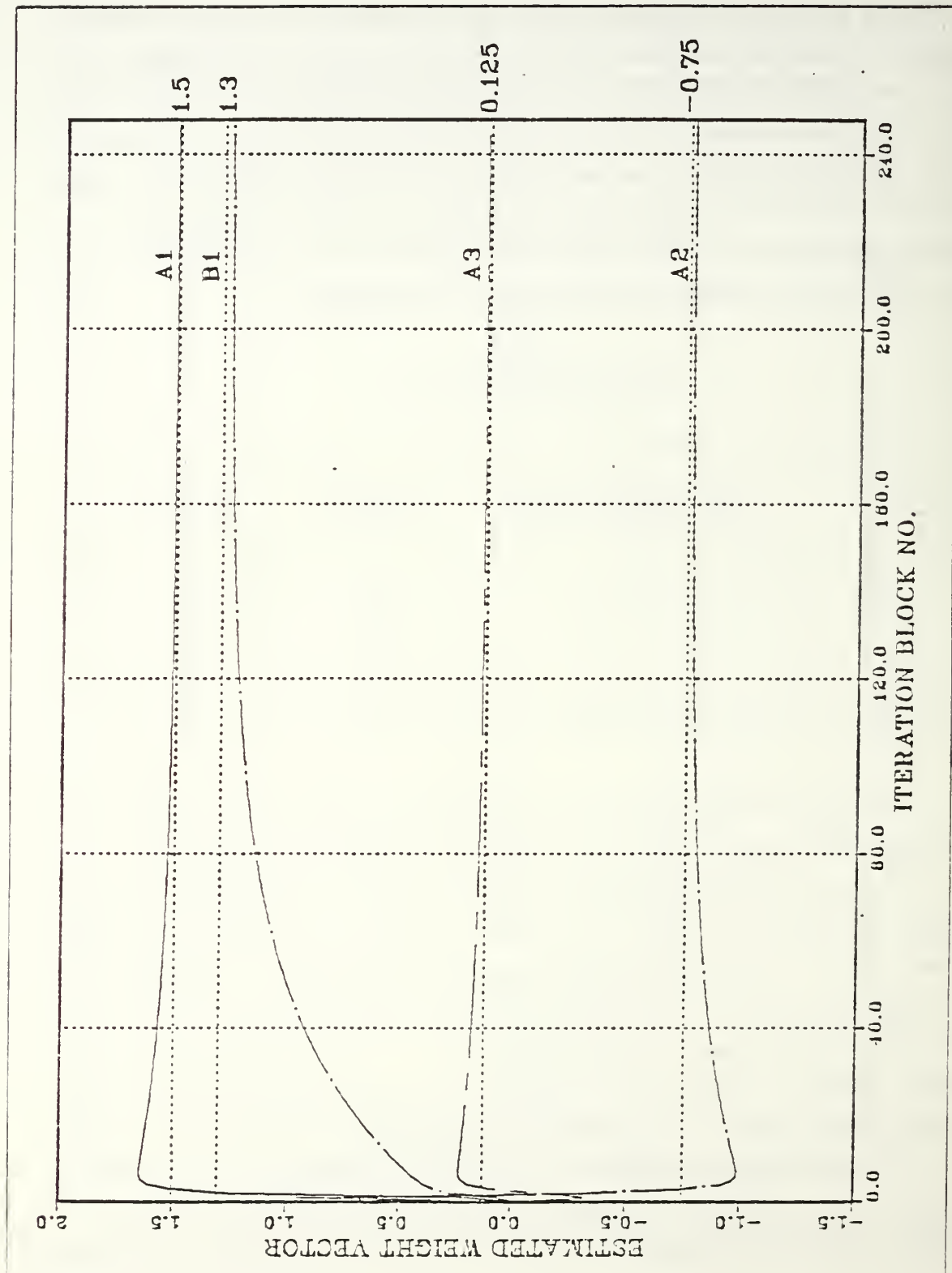


Figure 4.4 Example 4.2 - Parameter Estimation with $N = 5n$, W/O Noise.

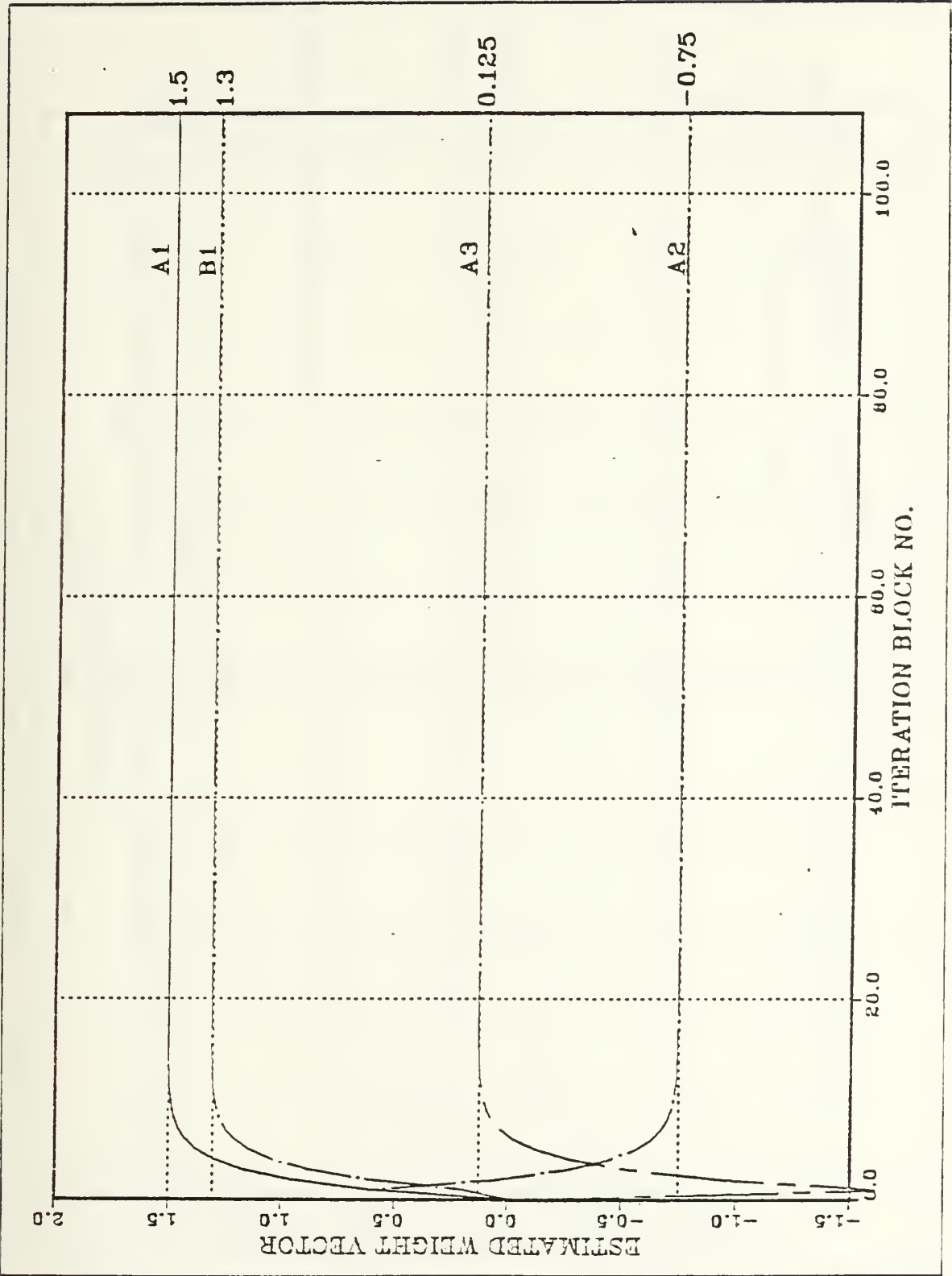


Figure 4.5 Example 4.2 - Parameter Estimation with $N = 15n$, W/O Noise.

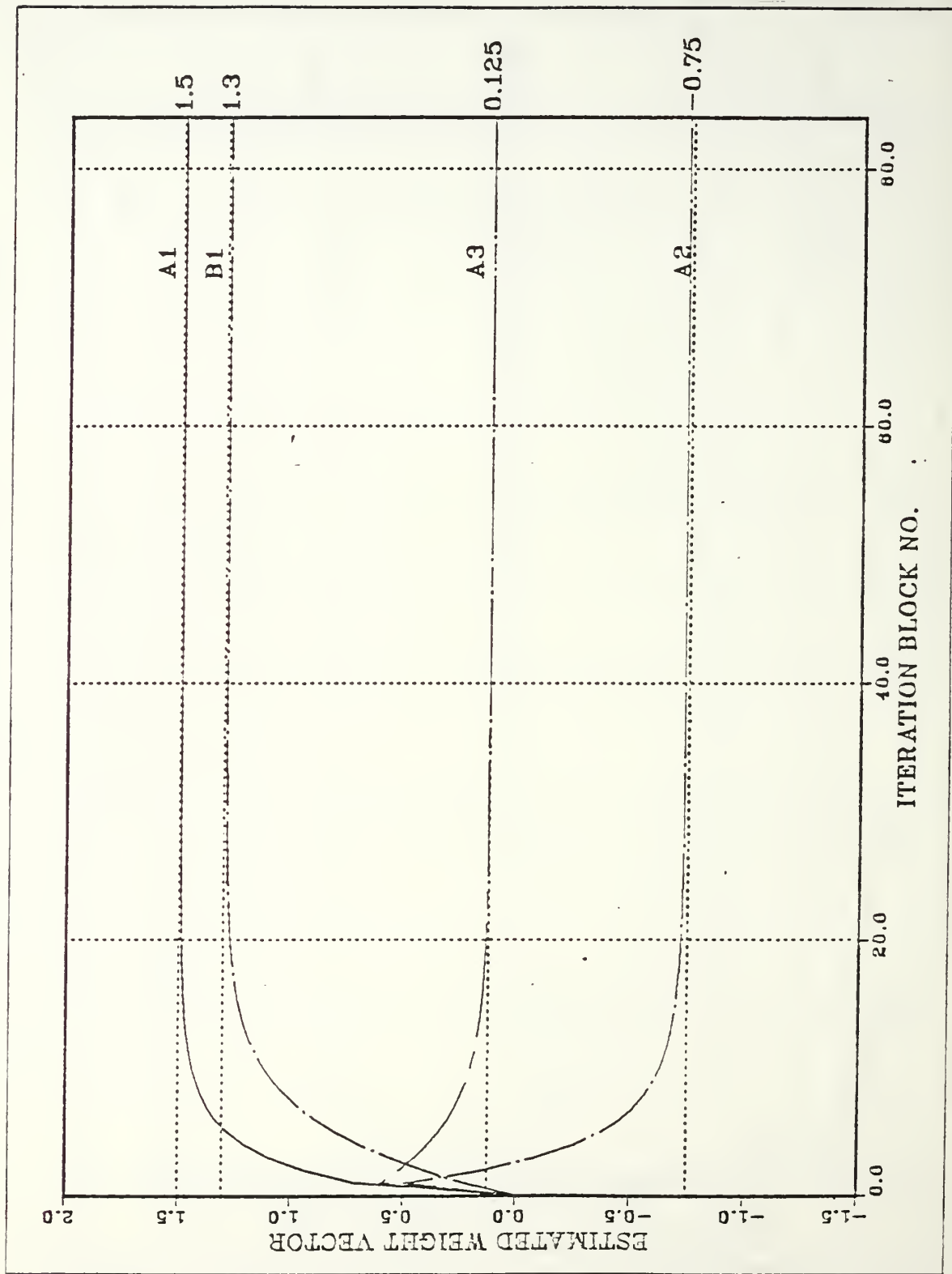


Figure 4.6 Example 4.2 - Parameter Estimation with $N = 20n$, W/O Noise.

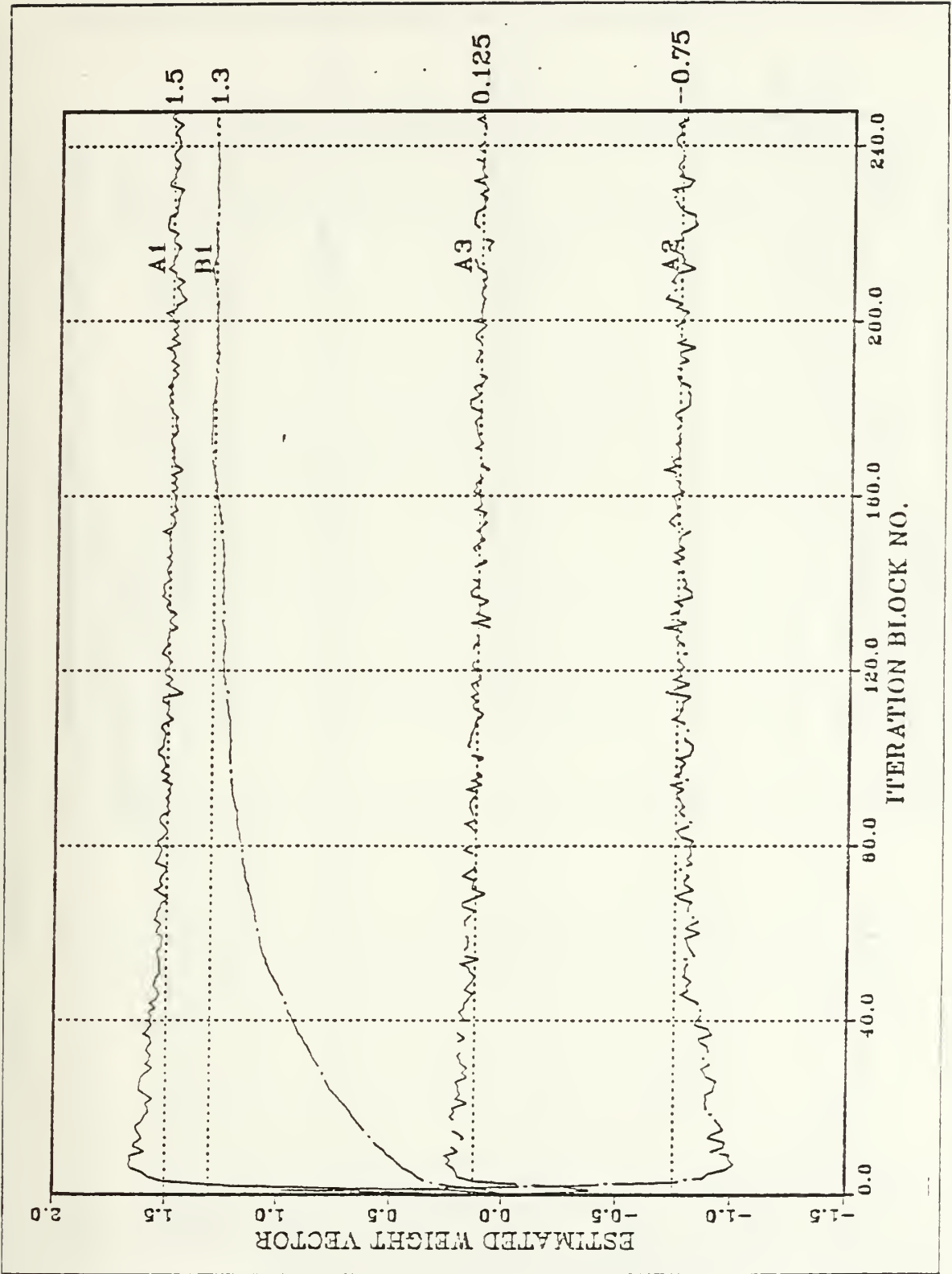


Figure 4.7 Example 4.2 - Parameter Estimation with $N = 5n$, $S/N = 10$.

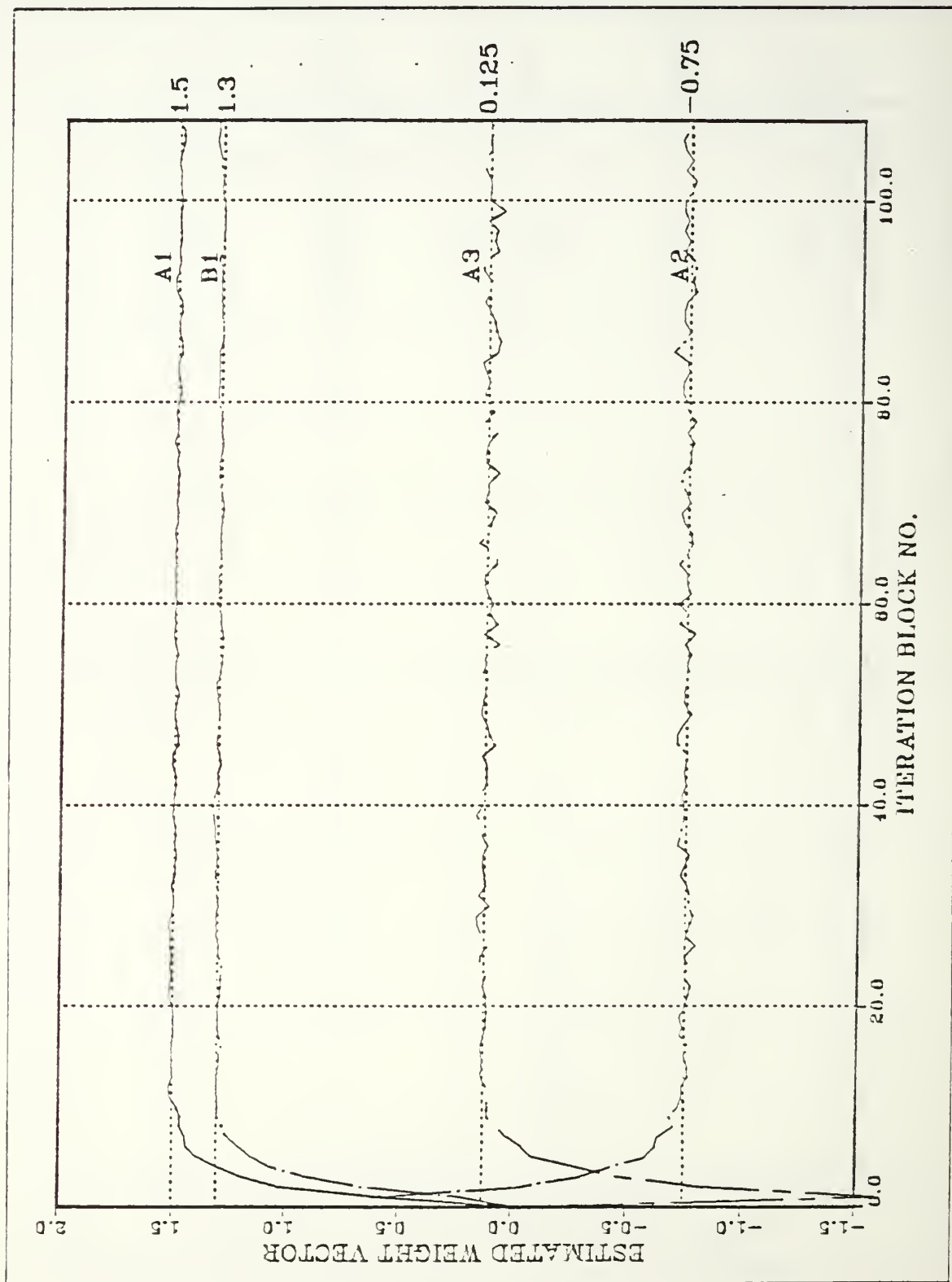


Figure 4.8 Example 4.2 - Parameter Estimation with $N = 15n$, $S/N = 10$.

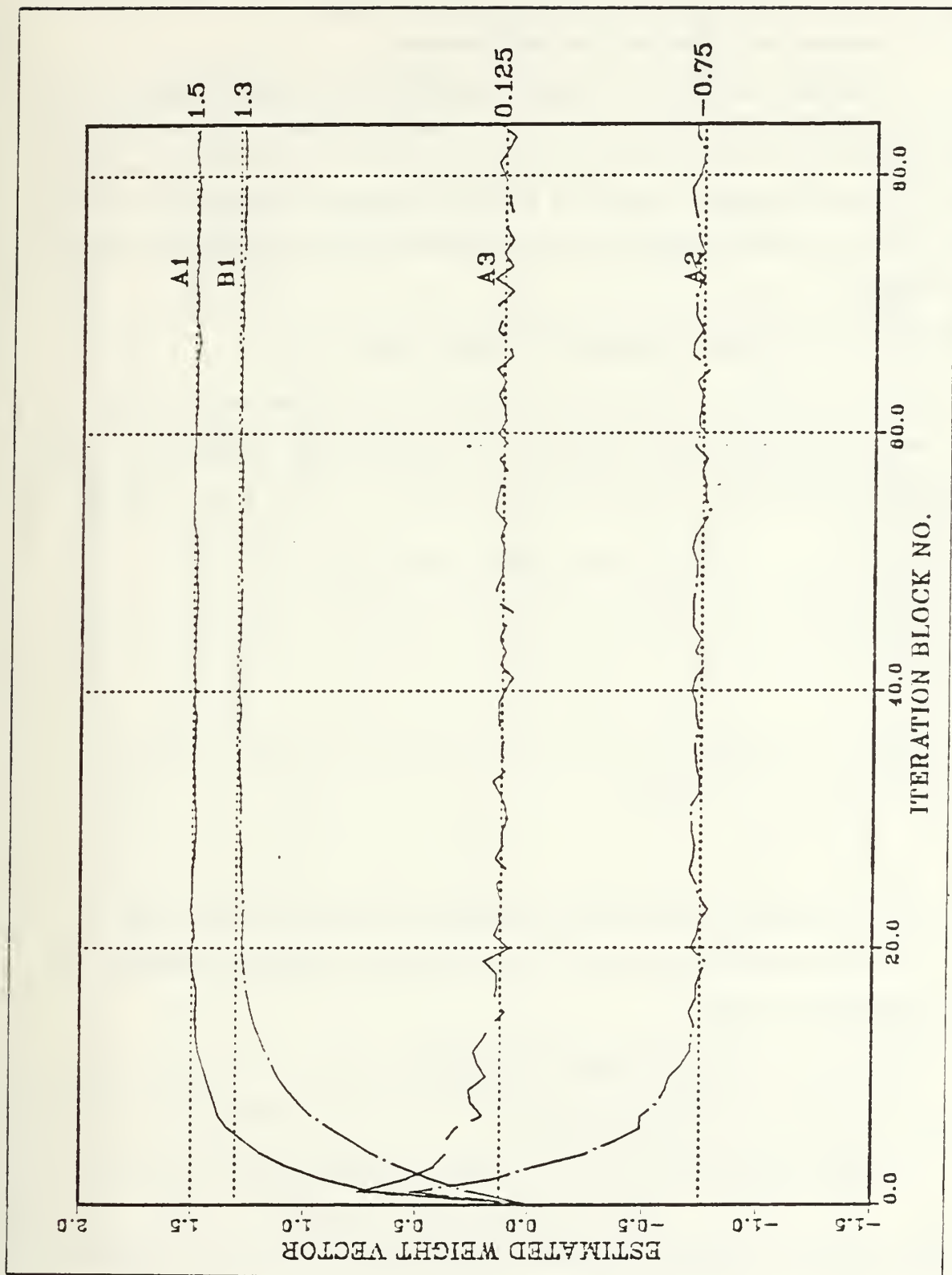


Figure 4.9 Example 4.2 - Parameter Estimation with $N = 20n$, $S/N = 10$.

be increased according to increasing of the block length, but if the block length exceed certain saddle point then the convergence rate starts to decrease. This optimal block length ($N = 15n$) is applied for the remaining simulation studies.

2. Application to Systems with Varying Parameters

A desirable application of adaptive identification is the tracking of time varying parameters. Then there is no obviously tradeoff between tracking ability and noise sensitivity [Ref. 3: p. 274] and it will be impossible to exactly follow parameters with a high rate of change. However, a slow time variation of the actual parameters can be often be tracked reasonably well. To illustrate this approaches consider the example below.

Example 4.3) (tracking of parameters with time-varying)

Consider the system with model as in equation (4.11) and time varying parameters. We simulated the input-output behaviour of the system for a number of block (represented by NB) = 150 using the parameter values

$$\begin{aligned} a_1 &= 1.8 \\ a_2 &= -1.08 & \text{if } 30 \leq \text{NB} \leq 60, 90 \leq \text{NB} \leq 120 \\ a_3 &= 0.216 \\ b_1 &= 1.0 \end{aligned}$$

$$\begin{aligned} a_1 &= 1.5 \\ a_2 &= -0.75 & \text{otherwise} \\ a_3 &= 0.125 \\ b_1 &= 1.3 \end{aligned}$$

The parameter estimates obtained with this example are displayed in Figure 4.10.

From the result, we can show that all estimated parameters converged very well to their expected values.

Example 4.4) (tracking of parameters with noise)

We take the two different types of model set for simulation in this example. For the case of equation error, the system is

$$y(t) = a_1 y(t-1) + a_2 y(t-2) + a_3 y(t-3) = b_1 u(t-1) + v(t) \quad (4.10)$$

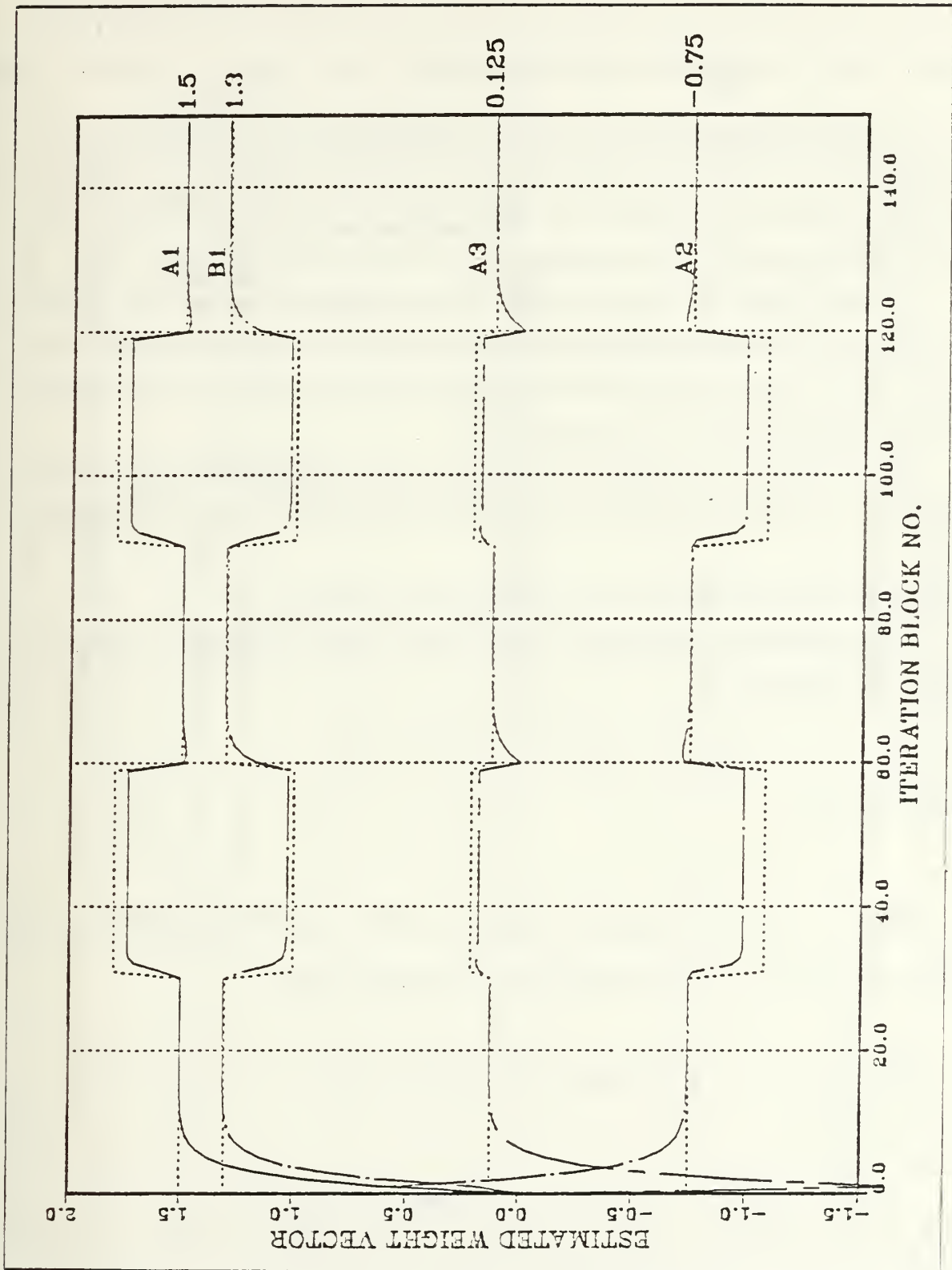


Figure 4.10 Example 4.3 - Tracking of Parameter with Time Varying, W/O Noise.

The other case for output error is

(4.11)

$$y(t) + a_1y(t-1) + a_2y(t-2) + a_3y(t-3) = b_1u(t-1) + v(t) + a_1v(t-1) + a_2v(t-2) + a_3v(t-3)$$

where $v(t)$ is white noise of zero mean for both systems. In this simulation, all parameters gain are used in equation (4.9) and tested with different input-signal to noise ratio, i.e., $S/N = 10$ and $S/N = 2$. The corresponding results are in Figures 4.11 and 4.12 for using equation error and Figures 4.13, 4.14 are from output error model.

In these results, the outputs of equation error exhibit less variations the results from output error. The comparison of the two model sets have not been carried out because there is no general result that clearly factors either model set. The amplitude of variation always depends on the system itself.

Final simulation is extended of Example 4.3 with two different types of noise model ($S/N = 10$) for the verification of parallel algorithm and illustrated in Example 4.5.

Example 4.5)(tracking with time-varying parameters and noise)

We simulated the combination of two system shown in Example 4.3 and 4.4. The systems are given by

$$y(t) + a_1y(t-1) + a_2y(t-2) + a_3y(t-3) = b_1u(t-1) + v(t)$$

for equation error,

and

$$y(t) + a_1y(t-1) + a_2y(t-2) + a_3y(t-3) = b_1u(t-1) + v(t) + a_1v(t-1) + a_2v(t-2) + a_3v(t-3)$$

for the output error case. The parameters above change as follows

$$a_1 = 1.8$$

$$a_2 = -1.08$$

$$a_3 = 0.216$$

$$b_1 = 1.0$$

$$30 \leq NB \leq 60, 90 \leq NB \leq 120$$

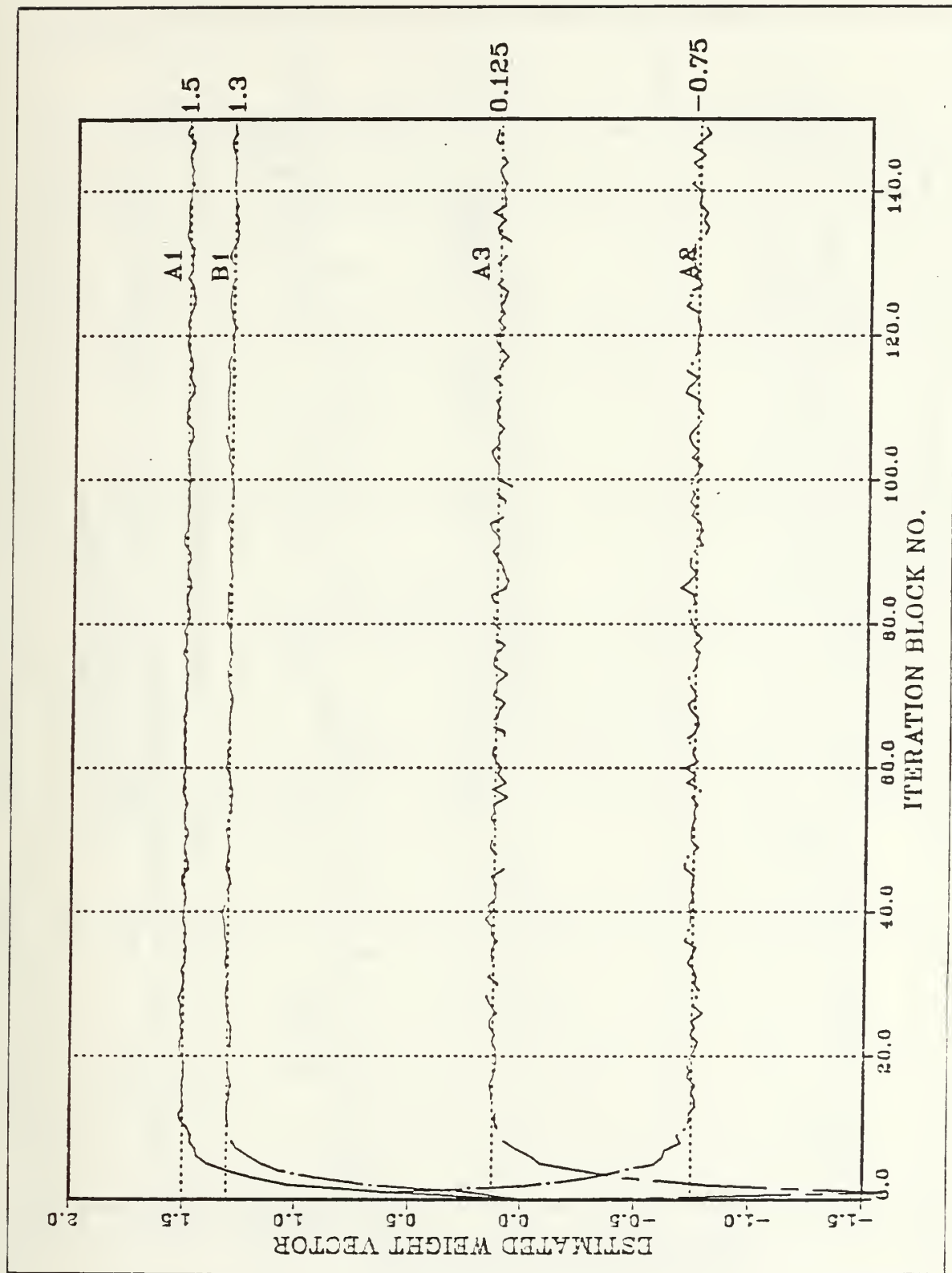


Figure 4.11 Example 4.4 - Tracking of Parameter with Equation Error, $S/N = 10$.

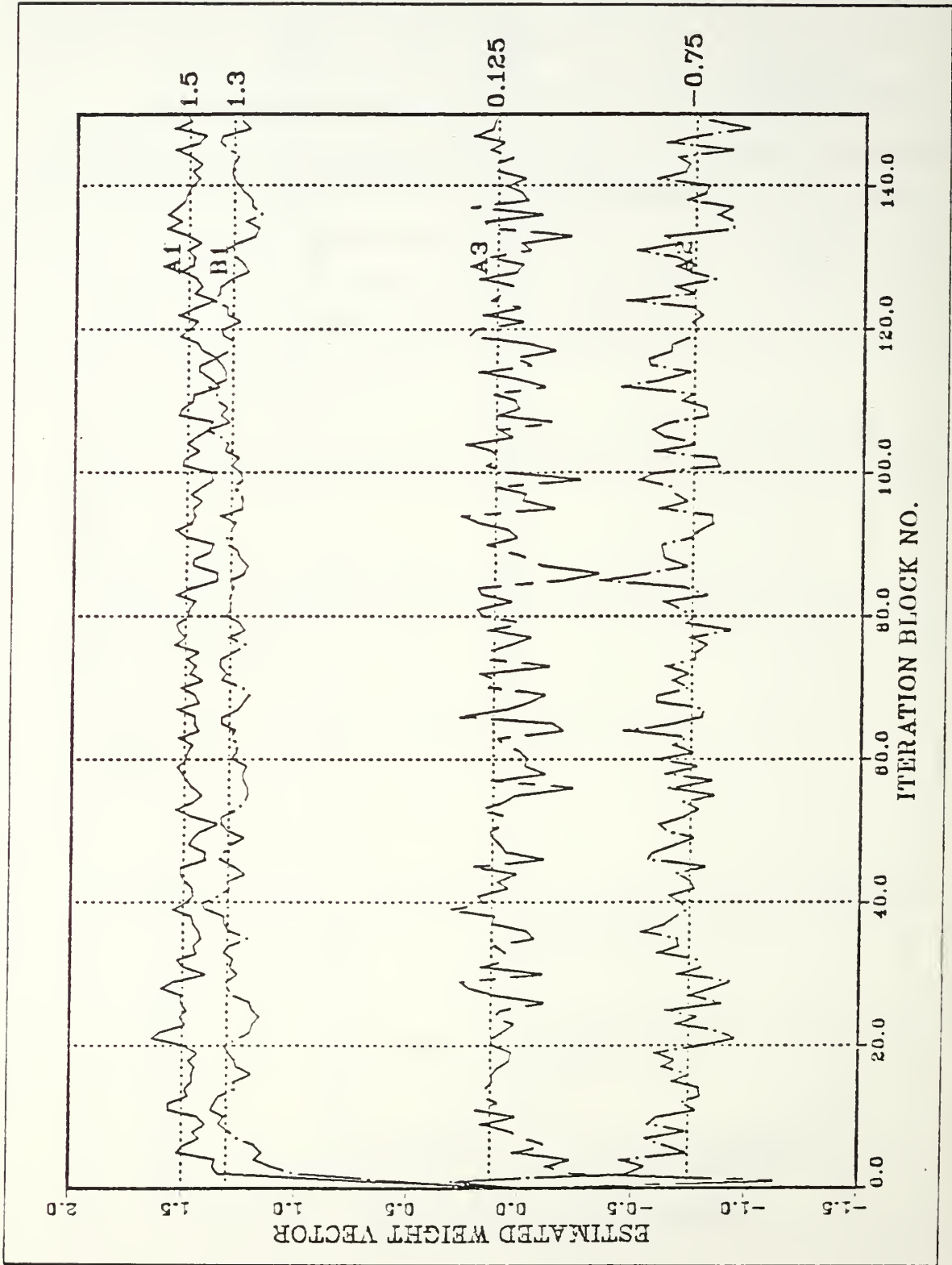


Figure 4.12 Example 4.4 - Tracking of Parameter with Equation Error, $S/N = 2$.

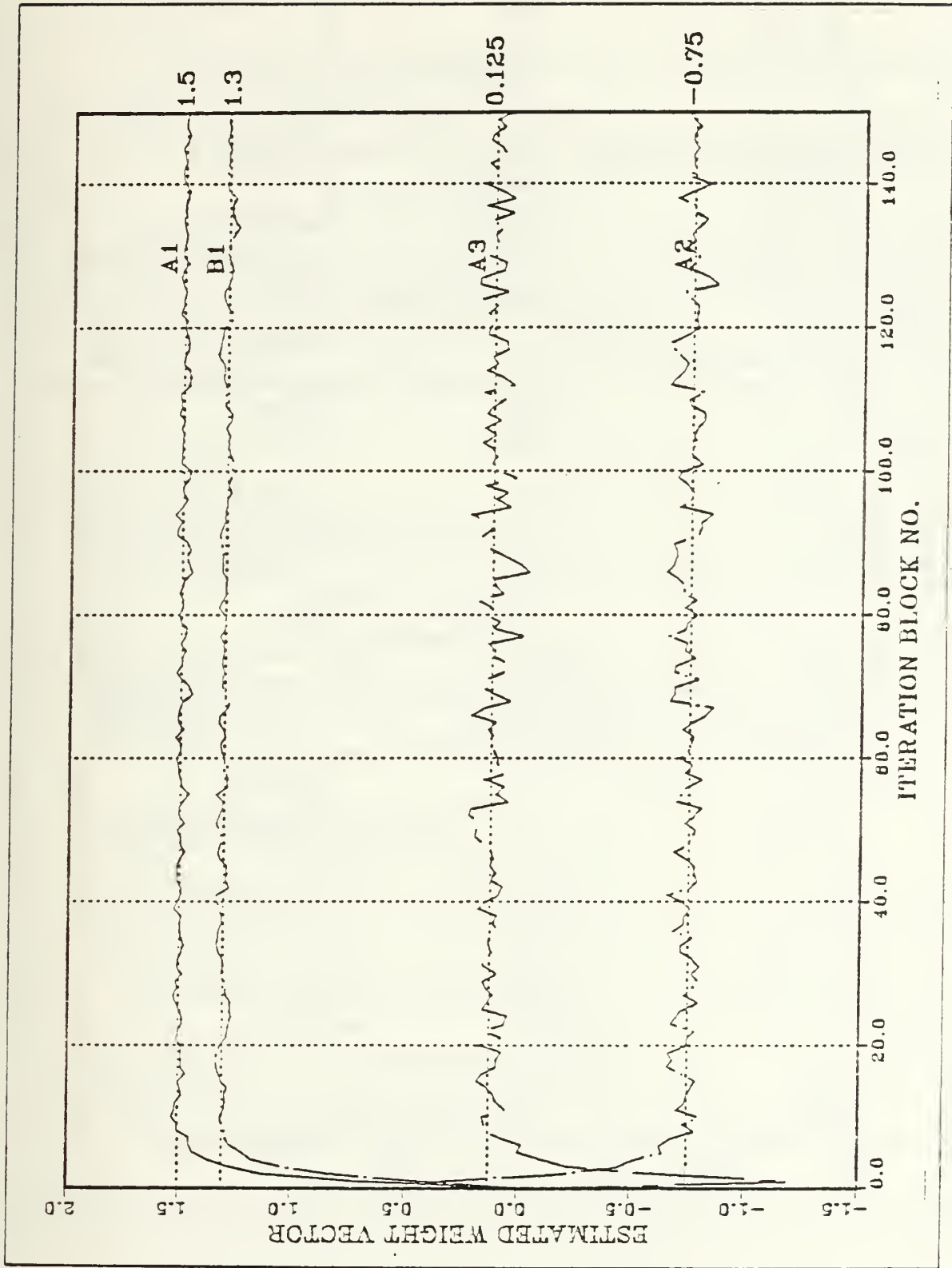


Figure 4.13 Example 4.4 - Tracking of Parameter with Output Error, $S/N = 10$.

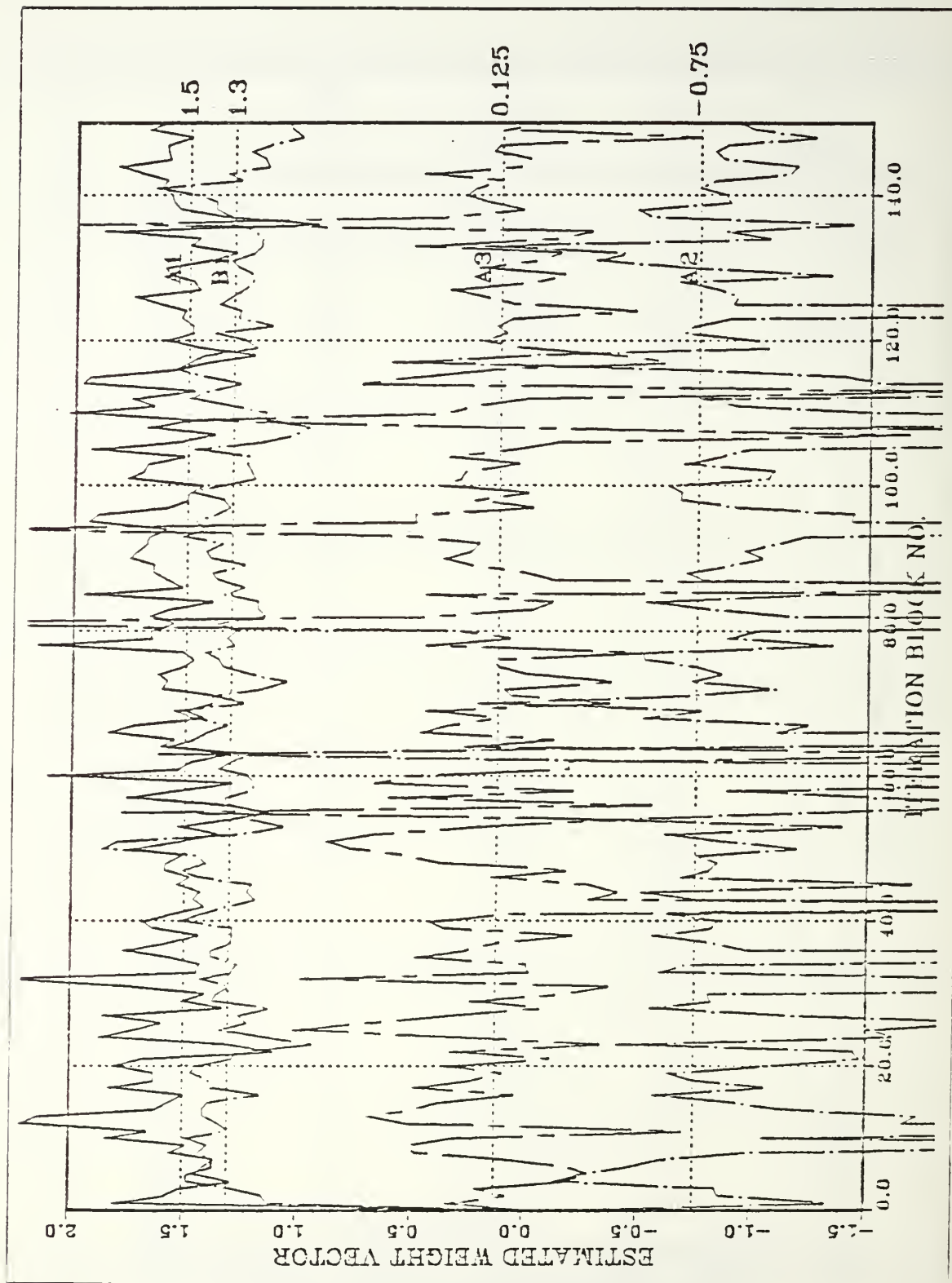


Figure 4.14 Example 4.4 - Tracking of Parameter with Output Error, $S/N = 2$.

$$\begin{aligned}
 a_1 &= 1.5 \\
 a_2 &= -0.75 && \text{otherwise} \\
 a_3 &= 0.125 \\
 b_1 &= 1.3
 \end{aligned}$$

The results are given in Figures in 4.15 and 4.16.

All parameters are converged very well under time-varying and noise condition.

in the discussion so far, we simulated several possible different conditions to investigate the behaviour of the parallel algorithm. The simulation study shows that we can obtain a reasonable parameter tracking in the presence of noisy measurements. The time block approach averages out the effects of random noise and provides an adjustable flexibility to improve the convergence rate of the parameter estimates.

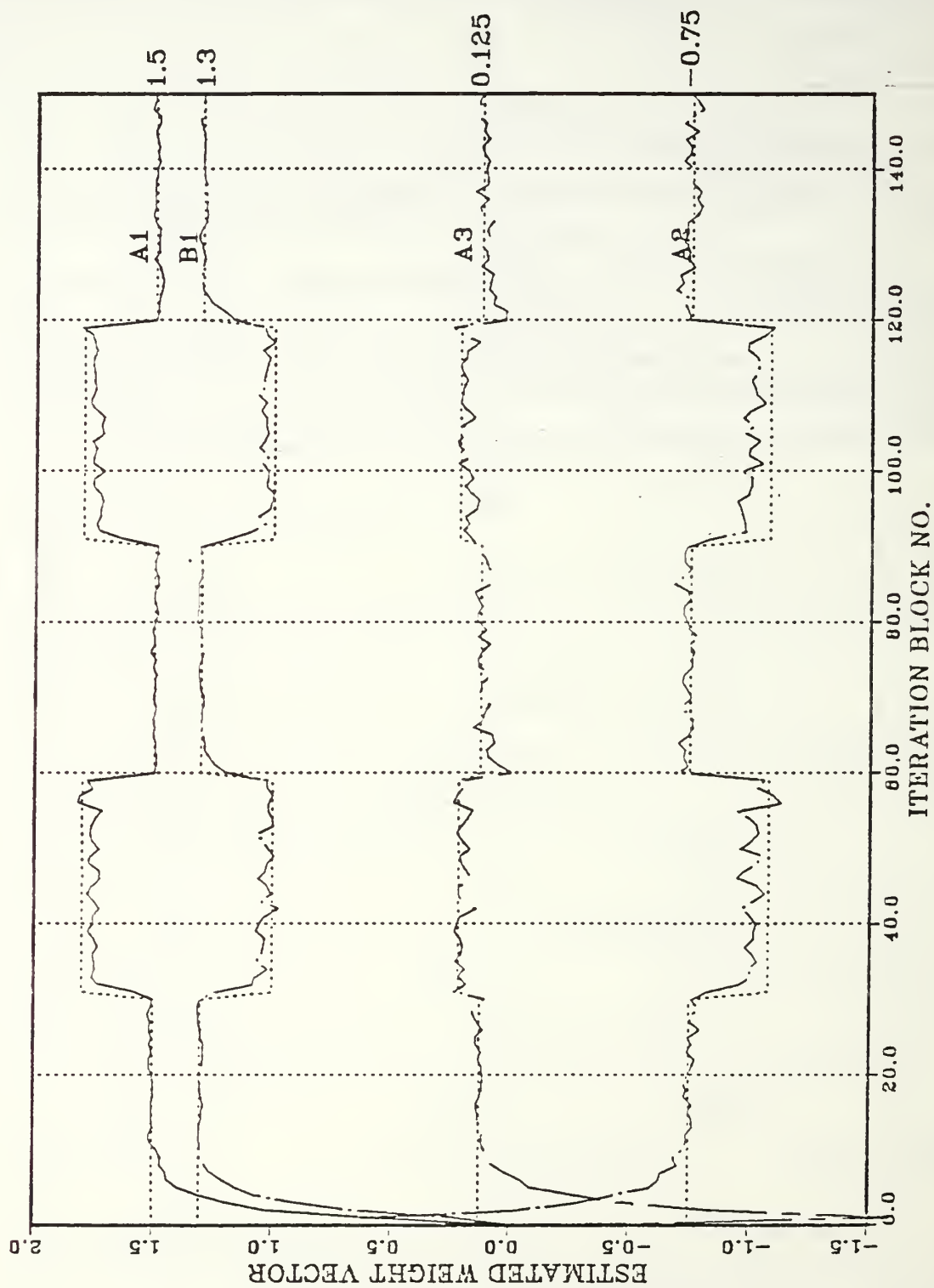


Figure 4.15 Example 4.5 - Parameter Tracking with Time-Varying, $S/N = 10$ of Equation Error.

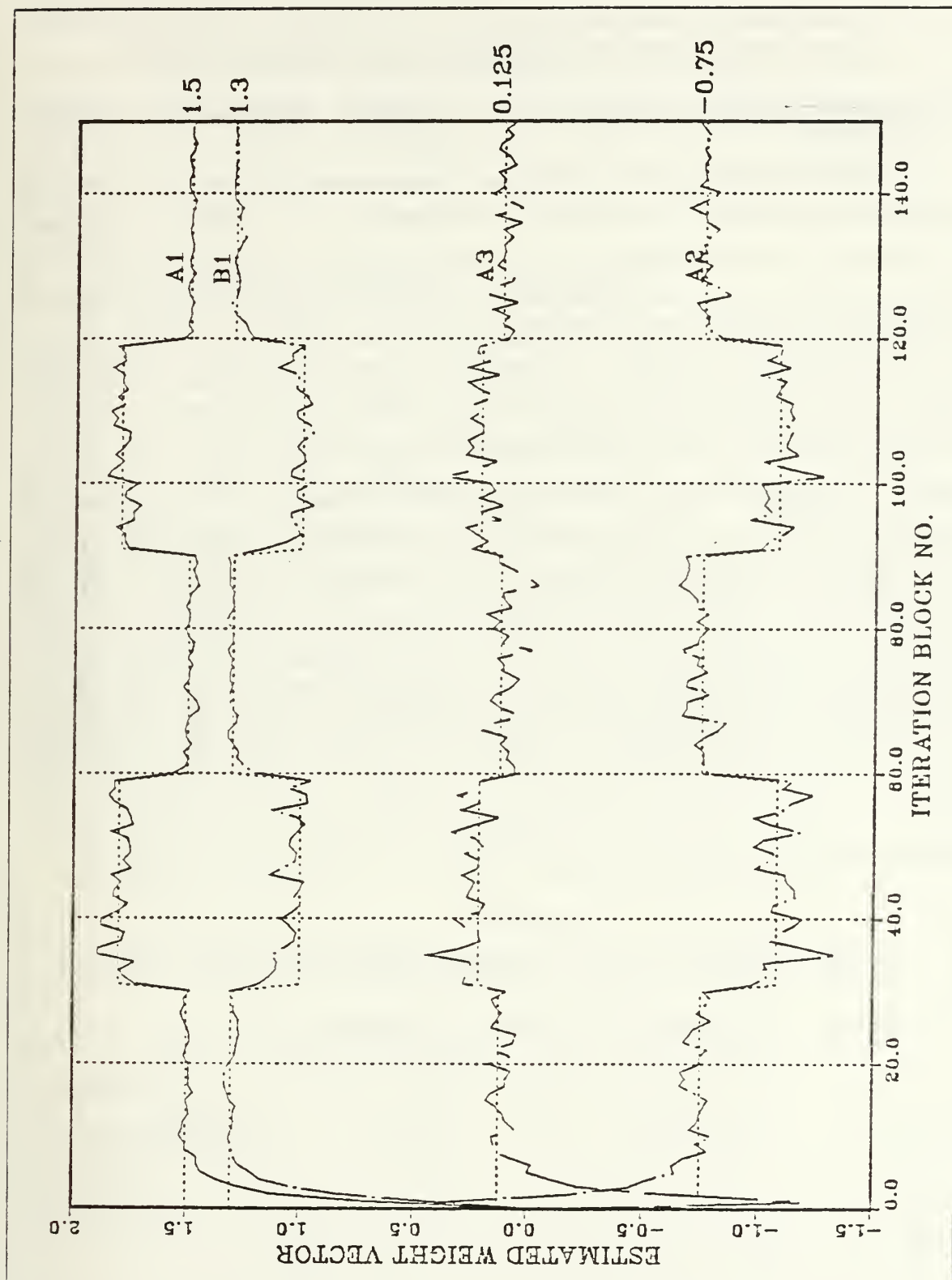


Figure 4.16 Example 4.5 - Parameter Tracking with Time-Varying, $S/N = 10$ of Output Error.

V. PERFORMANCE COMPARISON AND CONCLUSION

A. COMPUTATIONAL PROBLEM OF SERIAL RECURSIVE LEAST-SQUARES

The estimation of the model parameters using a parallel structure and standard least-squares algorithm has been described in Chapters II and III. We have discussed two possible ways of implementing the estimation algorithm: serial as in equation (2.13) and parallel as described in Chapter III. In this section we want to summarize the various trade-offs of the two algorithms, on the basis of their complexity in terms of computational power required. In particular consider the number of algebraic operations required for one iteration as a comparison of the two algorithms. For the case of serial implementation consider the two recursions in equation (2.13) which both update the covariance matrix as well as the parameter estimates. From a numerical standpoint the recursion of the covariance matrix $P(t)$ exhibits the highest complexity, since it requires a number of operations which encreases with d^2 , $d \times d$ being the dimensions of $P(t)$. Also it is worth mentioning that $P(t)$ is not updated by direct application of the recursion in (2.13) but it is rather computed by LUD factorization [Ref. 3: p. 336] in order to preserve its positive definiteness in the presence of numerical errors. In this way it is possible to compute the number of arithmetic operations required for matrix inversion and parameter estimation. Table 2 shows the total number of arithmetic operations for updating the parameters and also can be used to determine the time required per iteration.

TABLE 2
NUMBER OF ARITHMETIC OPERATION FOR UPDATING INPUT

	Matrix Inv.	Parameter Est.	Total
Addition	$1.5d^2 + 1.5d$	$d^2 + 2d$	$2.5d^2 + 3.5d$
Multiplication	$1.5d^2 + 5.5d$	$d^2 + 2d$	$2.5d^2 + 7.5d$
Division	d	d	$2d$
Subtraction		1	1

As shown in Table 2, for example, let us consider a 3rd order autoregressive model with three poles as in example 4.1. In this case the dimension of ϕ (represented by d) is 4, and a total 54 additions, 70 multiplications, 8 divisions and 1 subtraction are required for each parameter updating. It is clear that the sampling time, T_s increases with the square of $\phi(d)$ in the standard recursive least-squares. As a consequence of this we can see that the rate at which we can enter the data $\phi(t)$ into the algorithm is proportional to $1/d^2$.

B. COMPUTATION OF PARALLEL STRUCTURE

In the case of parallel structure, the computations required by the limitation in the maximum sampling frequency are performed in each cell at each clock cycle. As discussed in the previous chapters the estimation is performed by two distinct processors: P1 which performs matrix triangularization and operates within each time block, and P2 which solves the triangular system and it operates at the end of the time block. Each of these two processors has different complexities as shown below. The number of operations in each cell of the triangularization processor P1 is shown in Table 3. Since the cells operate in parallel, and in a pipeline fashion, at the end of each computing cycle we can enter new data, so the throughput (i.e., the rate at which we can feed new data into the processor) is constant and independent of the complexity of the system. We can say therefore that in this case the sampling time is of order $T_p = O(1)$, constant, compared with $T_p = O(d^2)$ in the recursive implementation.

TABLE 3
NUMBER OF ARITHMETIC OPERATION IN PARALLEL STRUCTURE

Section	Triangular Systolic		Linear Systolic	
Cell	Boundary	Internal	Boundary	Internal
Addition	1	2		1
Multiplication	3	4		1
Division	1			
Subtraction			1	

On the other hand, the serial implementation, the estimated parameters are available at any instant of time. This is not the case for the parallel algorithm presented since it has to wait until the end of the block to solve for the linear system of equation. Let us consider the time interval for parameters updating between both cases of the serial computation and parallel structure. This time is related to the amount of serial computation required by the two algorithm.

For example, let us consider the particular case shown Table 2 for serial computation, and assume the time block to be of length $N = 10n$ (n = number of order) in the parallel structure. Although we cannot say a priori which cell takes longer computing time, by referring to Table 3 we can conjecture that the internal and the boundary cells of triangular systolic array perform more complex operations than the linear array section. In Table 4, we compute the number of arithmetic operations to performed for each clock cycle in the serial computation and one entire time block ($N + 2d$) in the parallel structure shown in Figure 3.9. The former one based on either the number of operations in internal cells or boundary cells of the triangular systolic array.

TABLE 4
COMPARISON OF SERIAL AND PARALLEL STRUCTURE (DIM. Φ = 4)

	Serial Computation	Parallel Computation	
Basis		Boundary	Internal
Addition	54	38	76
Multiplication	70	104	152
Division	8	38	
Subtraction	1		

As shown in Table 4, it is clear that the number of required arithmetic operations in the serial computation per clock cycle is approximately half the number of operations needed for the entire time block ($N + 2d$) in the parallel structure. Since the number of operations is related with the time interval between parameters updating, we can have an idea of the rate at which we update the parameters. However we must

keep in mind that the serial algorithm updates the parameters on the basis of one sample only, while the parallel algorithm uses all samples within the time block. the benefit of this is a better performance of the parallel algorithm in case of noisy signals. This is in line with the fact discussed earlier that the sampling rate for the parallel algorithm can be much faster than the serial one, thus allowing more data to be processed in a given amount of time.

APPENDIX

COMPUTER PROGRAM OF PARALLEL ALGORITHM

PURPOSE

This program calculated the estimated weight vector of least-squares using parallel algorithm, the input is a coefficient of polynomial ; the form

$$H(z) = \frac{B_1 z^{-1}}{1 + A_1 z^{-1} + A_2 z^{-2} + A_3 z^{-3}}$$

where the all coefficients are given interactively and real. This program is limited with system model as shown above transfer function, same type of denominator with 1st, 2nd, and 3rd order cases and one input parameter. If run this program, should be extended virtual storage capacity to 2M in advance due to the large array. This program consists of main and four different subroutines which are performed each cell's function.

VARIABLE DEFINITION

```

REAL A(300,5,300),U(300,5,300),XIN(5,5),CINP(5,5),SINP(5,5)
REAL COUP(5,5),SOUP(5,5),SOUN(5,5),CINN(5,5),SINN(5,5),XOUT(5,5)
REAL SX(5,5),A1,B1,YNP,YP,INUP,SIG,PI,COUN(5,5),EAL,EB1,NSX(5,5)
REAL A2,YPM1,ABB,EA2,YPM,XA1,KB1,PI10,ZIN(5,5),ZOUT(5,5)
REAL WK(5,5),REB(12),AM,V,S,DIN(6,6,15),DOUT(7,7,15)
REAL XLIN(7,7,15),WKIN(5,5),WK1(5,5),KIB
INTEGER K,L,M,NT,IK,IL,NBN,K1,L1,N,IT,NI,IX,NM1,NB,K2,L2,KR,LR,NC

```

***** INITIALIZE THE GIVEN TRANSFER FUNCTION AND ALL VARIABLES *****

```

1000 PRINT *, 'ENTER THE SIZE OF MATRIX N BY N ** N ?'
    READ *, N
    PRINT *, 'N = ', N
    PRINT *, ' '
    PRINT *, 'ENTER THE ORDERS OF DEN. OF DIFF. EQ.**NO ?'
    READ *, NO
    PRINT *, 'NO = ', NO
    PRINT *, ' '
    PRINT *, 'HOW MANY BLOCK DO YOU WANT TO ITERATE **NBN ?'
    READ *, NBN
    PRINT *, 'NBN = ', NBN
    IF ( NO .EQ. 1) THEN
    PRINT *, 'FORMAT OF 1ST ORDER DIFF.EQ.**Y(Z) = B1*U(Z-1) + A1*Y(Z-
*1)'
    PRINT *, 'ENTER THE COEFF. OF A1 ?'
    READ *, A1
    PRINT *, 'A1 = ', A1
    PRINT *, 'ENTER THE COEFF. OF B1 ?'

```

```

READ * ,B1
PRINT *, 'B1 = ',B1
WRITE (6,700)
ELSEIF ( NO .EQ. 2) THEN
PRINT *, 'FORMAT OF 2ND ORDER DIFF.EQ.***Y(Z) = B1*U(Z-1) + A1*Y(Z-
*1)- A2*Y(Z-2)'
PRINT *, 'ENTER THE COEFF. OF A1 ?'
READ * ,A1
PRINT *, 'A1 = ',A1
PRINT *, 'ENTER THE COEFF. OF A2 ?'
READ * ,A2
PRINT *, 'A2 = ',A2
PRINT *, 'ENTER THE COEFF. OF B1 ?'
READ * ,B1
PRINT *, 'B1 = ',B1
WRITE (6,720)
ELSEIF ( NO .EQ. 3) THEN
PRINT *, 'FORMAT OF 3RD ORDER DIFF.EQ.***Y(Z) = B1*U(Z-1) + A1*Y(Z-
*1)- A2*Y(Z-2) + A3*Y(Z-3)'
PRINT *, 'ENTER THE COEFF. OF A1 ?'
READ * ,A1
PRINT *, 'A1 = ',A1
PRINT *, 'ENTER THE COEFF. OF A2 ?'
READ * ,A2
PRINT *, 'A2 = ',A2
PRINT *, 'ENTER THE COEFF. OF A3 ?'
READ * ,A3
PRINT *, 'A3 = ',A3
PRINT *, 'ENTER THE COEFF. OF B1 ?'
READ * ,B1
PRINT *, 'B1 = ',B1
WRITE (6,740)
ELSE
PRINT *, 'ERROR ** PLEASE TRY AGAIN AMONG THOSE 1,2,3 FOR NO'
GOTO 1000
END IF
NM1=N-1
NB =15*NO
SIG = 1.
PI = 3.141592
PI10 = PI/10
IX = 1
S = 0.07071
AM = 0.
YP = 0.
YPM = 0.
YPM1 = 0.
VM1 = 0
VM2 = 0
VM3 = 0
INUP = COS(PI10) + COS(PI10*2) + COS(PI10*5)+ COS(PI10*6)

```

***** MAIN PROGRAM BEGIN *****

```

DO 10 IK = 1,N
DO 20 IL = 1,N
IF ((IK .EQ. IL ) .AND. (IK .LT. N))THEN
COUP(IK,IK)=1.
SOUP(IK,IK)=0.
SX(IK,IL) = SIG
ELSE
SX(IK,IL) = 0.
END IF
IF (IK .EQ. N) THEN
ZIN(IK,IL) = 0
WK(IK,IL) = 0
END IF
IF (K .NE. 1) THEN
XIN(IK,IL)=0.

```

```

      END IF
20  CONTINUE
10  CONTINUE

```

```

      DO 330 IB = 1,NBN
      DO 30 NT = 1,NB

```

***** TIME VARYING INPUT *****

```

C      IF (((IB.GT.30).AND.(IB.LT.60)).OR.((IB.GT.90).AND.(IB.LT.120)))
C      * THEN
C          A1 = 1.8
C          A2 = 1.08
C          A3 = 0.216
C          B1 = 1.0
C      ELSE
C          A1 = 1.5
C          A2 = 0.75
C          A3 = 0.125
C          B1 = 1.3
C      END IF
      IF (NT .LE. NB-N+1) THEN
          CALL GAUSS(IX,S,AM,V)
          IF (NO .EQ. 1) THEN
              YNP = B11*INUP - A11*YP
C          YNP = B1*INUP + A1*(YP+V)
              JF = 1
              A(NT,JF,NT) = YP
              A(NT,JF+1,NT) = INUP
              A(NT,JF+2,NT) = YNP
              ELSEIF ( NO .EQ. 2) THEN
C          YNP = B1*INUP + A1*(YP+V)- A2*YPM
C          YNP = B1*(INUP+V) - A1*YP
              YNP = B1*INUP + A1*YP-A2*YPM
              JF = 1
              A(NT,JF,NT) = YPM
              A(NT,JF+1,NT) = YP
              A(NT,JF+2,NT) = INUP
              A(NT,JF+3,NT) = YNP
              ELSEIF ( NO .EQ. 3) THEN
C          YNP = B1*INUP + A1*YP- A2*YPM+A3*YPM1+V+A1*VM1+A2*VM2+A3*VM3
              YNP = B1*INUP +A1*YP- A2*YPM + A3*YPM1 +V
C          YNP = B1*INUP + A1*YP-A2*YPM+ A3*YPM1
              JF = 1
              A(NT,JF,NT) = YPM1
              A(NT,JF+1,NT) = YPM
              A(NT,JF+2,NT) = YP
              A(NT,JF+3,NT) = INUP
              A(NT,JF+4,NT) = YNP
          END IF

```

***** TRANSFORMATION OF INPUT DATA WITH SKEW ORDER *****

```

      DO 40 J = 1,N
      IT = J-1
      NI = NT+IT
      U(NI,J,NI) = A(NT,J,NT)
40  CONTINUE
      END IF

```

***** CALCULATE THE ORTHOGONAL TRIANGULAR MATRIX BY GIVENS
rotation AND INITIALIZE THE ORTHOGONAL MATRIX *****

***** CALCULATE THE BOUNDARY AND INTERNAL CELLS *****

```

      DO 50 K = 1,N-1
      DO 60 L = 1,N
      IF ( NT .LT. N ) THEN

```

```

CINP(K,L+1) = COUP(K,L)
SINP(K,L+1) = SOUP(K,L)
SX(K,L) = SX(K,L)
END IF
IF ((K.LE.L) .AND. (K+L .LE. NT+1)) THEN
IF (K .EQ. 1) THEN
XIN(K,L) = U(NT,L,NT)
END IF

```

***** CALCULATE THE BOUNDARY CELL *****

```

IF ( K .EQ. L ) THEN
CALL BND (K,L,NT,XIN,SX,COUP,SOUP)
CINN(K,L+1) = COUP(K,L)
SINN(K,L+1) = SOUP(K,L)
IF (NT .EQ. NB) THEN
CINN(K,L+1) = 0.
SINN(K,L+1) = -1.
END IF

```

***** CALCULATE THE INTERNAL CELL ****

```

ELSEIF(K .NE. L) THEN
CALL INRN (K,L,NT,XIN,XOUT,SX,NSX,CINP,SINP)
CINN(K,L+1) = CINP(K,L)
SINN(K,L+1) = SINP(K,L)
SX(K,L) = NSX(K,L)
END IF
ELSEIF ( K .GT. L ) THEN
SX(K,L)=0.
END IF
60 CONTINUE
50 CONTINUE
DO 70 K1 = 1,N-1
DO 80 L1 = 1,N
IF ( K1 .NE. L1) THEN
CINP(K1,L1) = CINN(K1,L1)
SINP(K1,L1) = SINN(K1,L1)
END IF
XIN(K1+1,L1) = XOUT(K1,L1)
80 CONTINUE
70 CONTINUE
IF ( NO .EQ. 1) THEN
YP = YNP
ELSEIF ( NO .EQ. 2) THEN
YPM = YP
YP = YNP
ELSEIF ( NO .EQ. 3) THEN
YPM1 = YPM
YPM = YP
YP = YNP
VM3 = VM2
VM2 = VM1
VM1 = V
END IF
NX = NT
INUP = COS(PI10*NX)+COS(PI10*NX*2)+COS(PI10*NX*5)+COS(PI10*6*NX)

```

***** CALCULATE THE ESTIMATED COEFFICIENT WITH RESET *****

```

NC = 1
IF (NT .EQ. NB ) THEN
DO 600 IM = 1,N
DO 610 JM = 1,N
XOUT(IM,JM) = 0.
XIN(IM,JM) = 0.
610 CONTINUE
600 CONTINUE
DO 400 NR = 1,2*N-1

```

```

DO 410 KR = 1,N
DO 420 LR = 1,N
IF((KR .NE. LR) .AND. (KR .LT. N)) THEN

CALL INRN (KR,LR,NR,XIN,XOUT,SX,NSX,CINP,SINP)
*****
IF (LR .NE. N ) THEN
DIN(KR,LR,NR) = XOUT(KR,LR)
END IF
CINN(KR,LR+1) = CINP(KR,LR)
SINN(KR,LR+1) = SINP(KR,LR)
SX(KR,LR) = NSX(KR,LR)
ELSEIF ((KR .EQ. LR) .AND. (KR .LT. N)) THEN
XIN(KR,LR) = 0.

CALL BND(KR,LR,NR,XIN,SX,COUP,SOUP)
*****
CINN(KR,LR+1) = COUP(KR,LR)
SINN(KR,LR+1) = SOUP(KR,LR)
END IF
IF(KR .EQ. N) THEN
IF (LR .EQ. N) THEN

CALL LINBD (N,KR,LR,NR,ZIN,XIN,WK,WK1,SX,REB,NC)
*****
IF ((NR.EQ. 3).OR.(NR.EQ.5).OR.(NR .EQ. 7).OR.(NR .EQ.9)) THEN
END IF
ELSEIF ((LR .GT. 1) .AND. (LR .LT. N)) THEN

CALL LINNT(KR,LR,NR,ZIN,XLIN,WK,WKIN,ZOUT)
*****
END IF
END IF
420 CONTINUE
410 CONTINUE
DO 430 K2 = 1,N
DO 440 L2 = 1,N
IF (( K2 .NE. L2 ) .AND. (K2 .LT. N)) THEN
CINP(K2,L2) = CINN(K2,L2)
SINP(K2,L2) = SINN(K2,L2)
END IF
IF (L2 .LT. N-1 ) THEN
XIN(K2+1,L2+1) = DOUT(K2,L2,NR)
DOUT(K2,L2,NR+1) = DIN(K2,L2,NR)
ELSEIF (L2 .EQ. N-1) THEN
XLIN(N,K2+1,NR+2) = DIN(K2,L2,NR)
ELSEIF (L2 .EQ. N) THEN
XIN(K2+1,L2) = XOUT(K2,L2)
END IF
IF (K2 .EQ. N) THEN
ZIN(K2,L2+1) = ZOUT(K2,L2)
IF (L2 .EQ. N) THEN
WKIN(K2,L2-1) = WK1(K2,L2)
ELSE
WKIN(K2,L2-1) = WK(K2,L2)
END IF
END IF
440 CONTINUE
430 CONTINUE
400 CONTINUE

***** REINITIALIZE FOR NEW BLOCK PROCESSING *****

DO 470 N3 = 1,3*N-3
DO 450 K3 = 1,N
DO 460 L3 = 1,N
IF (K3 .NE. 1) THEN
XIN(K3,L3) = 0.
END IF

```

```

XLIN(K3,L3,N3) = 0.
DIN(K3,L3,N3) = 0.
DOU(K3,L3,N3) = 0.
IF (K3 .LT. N) THEN
  CINP(K3,L3) = 1.
  SINP(K3,L3) = 0.
  IF (K3 .EQ. L3) THEN
    SX(K3,L3) = SIG
  ELSE
    SX(K3,L3) = 0.
  END IF
ELSEIF (K3 .EQ. N) THEN
  WK(K3,L3) = 0.
  ZIN(K3,L3) = 0.
  END IF
460 CONTINUE
450 CONTINUE
470 CONTINUE
  XIB = IB
  IF (NO .EQ. 1) THEN
    WRITE (6,710) IB,REB(5),REB(3)
    SX(1,3) = REB(5)*SIG
    SX(2,3) = REB(3)*SIG
  ELSEIF (NO .EQ. 2) THEN
    WRITE (6,730) IB,REB(5),REB(7),REB(3)
    SX(1,4) = REB(7)*SIG
    SX(2,4) = REB(5)*SIG
    SX(3,4) = REB(3)*SIG
  ELSEIF (NO .EQ. 3) THEN
    WRITE (6,750) XIB,REB(5),REB(7),REB(9),REB(3)
    SX(1,5) = REB(9)*SIG
    SX(2,5) = REB(7)*SIG
    SX(3,5) = REB(5)*SIG
    SX(4,5) = REB(3)*SIG
  END IF
  END IF
  30 CONTINUE
  330 CONTINUE
  STOP
700 FORMAT ('1',3X,'BLOCK NO.',6X,'COEFF. A1',9X,'COEFF. B1')
710 FORMAT (3X,I4,11X,2(F10.7,7X))
720 FORMAT ('1',3X,'BLOCK NO.',6X,'COEFF. A1',9X,'COEFF. A2',9X,'COEFF
*. B1')
730 FORMAT (3X,I4,12X,3(F10.7,7X))
740 FORMAT ('1',3X,'BLOCK NO.',6X,'COEFF. A1',9X,'COEFF. A2',9X,'COEF
*F. A3',9X,'COEFF. B1')
750 FORMAT (3X,F4.0,12X,4(F10.7,7X))
END

```

***** SUBROUTINE GROUP FOR CELL'S FUNCTION *****

```

SUBROUTINE BND (K,L,NT,XIN,SX,COUP,SOUP)
*****
REAL XIN(5,5),SX(5,5),COUP(5,5),SOUP(5,5)
REAL NSQ
INTEGER K,L,NT
IF (XIN(K,L) .EQ. 0) THEN
  COUP(K,L)=1.
  SOUP(K,L)=0.
  SX(K,L) = SX(K,L)
ELSE
  NSQ = SX(K,L)**2+XIN(K,L)**2
  COUP(K,L)=SX(K,L)/NSQ
  SOUP(K,L)=XIN(K,L)/NSQ
  SX(K,L)=1.
END IF

```

```

RETURN
END

```

```

SUBROUTINE INRN (K,L,NT,XIN,XOUT,SX,NSX,CINP,SINP)
*****
REAL XIN(5,5),SX(5,5),CINP(5,5),SINP(5,5),XOUT(5,5),NSX(5,5)
INTEGER K,L,NT
XOUT(K,L)=-SINP(K,L)*SX(K,L)+CINP(K,L)*XIN(K,L)
NSX(K,L)=CINP(K,L)*SX(K,L)+SINP(K,L)*XIN(K,L)
RETURN
END

```

```

SUBROUTINE LINNT(KR,LR,NR,ZIN,XLIN,WK,WKIN,ZOUT)
*****
REAL ZIN(5,5),XLIN(7,7,15),WK(5,5),ZOUT(5,5),WKIN(5,5)
INTEGER KR,LR,NR
ZOUT(KR,LR) = ZIN(KR,LR) + XLIN(KR,LR,NR)*WKIN(KR,LR)
WK(KR,LR) = WKIN(KR,LR)
RETURN
END

```

```

SUBROUTINE LINBD(N,KR,LR,NR,ZIN,XIN,WK,WK1,SX,REB,NC)
*****
REAL ZIN(5,5),XIN(5,5),WK(5,5),WK1(5,5),SX(5,5),REB(5)
INTEGER KR,LR,NR,N,NC
WK(KR,LR) = SX(KR,LR)
WK1(KR,LR) = XIN(KR,LR) - ZIN(KR,LR)
SX(KR,LR) = WK1(KR,LR)
REB(NR) = WK(KR,LR)
RETURN
END

```

LIST OF REFERENCES

1. Goodwin G.C. and Sin K.S., *Adaptive Filtering, Prediction and Control*, Prentice-Hall, INC, 1984.
2. Anderson B.D.O. and Johnson C.R. Jr., "Exponential Convergence of Adaptive Identification and Control Algorithm", *Automatica*, 18.1, 1982.
3. Ljung L. and Soderstorm T., *Theory and Practice of Recursive Identification*, The MIT press, 1986.
4. Plackett R.L. "Some Theorems in Least Squares", *Biometrika*, Vol. 37, 1950.
5. Astrom K.J. and Eykhoff P., "System Identification - A Survey", *Automatica*, Vol. 7, 1971.
6. Kung S.Y. and Juhl J.T., *VLSI for Pattern Recognition and Image Processing*, Springer Verlag, 1984.
7. Hammarling S., "A Note on Modification to the Givens Plane Rotation", *J. Inst Maths Applies*, Vol. 13, 1975.
8. Duff Lain S. and Stewart G.W., *Sparse Matrix Proceedings*, Siam, 1978.
9. Gentleman W.H. and Kung H.T., "Real time Signal Processing IV". *SPIE*, 1981.
10. Simon Hykin, *Adaptive Filter Theory*, Prentice-Hall, 1986.

INITIAL DISTRIBUTION LIST

	No. Copies
1. Library, Code 0142 Naval Postgraduate School Monterey, California 93943-5002	2
2. Defence Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
3. Department Chairman, Code 62 Dept. of Electrical and Computer Engineering Naval Postgraduate School Monterey, California 93943-5000	1
4. Prof. Robert Cristi, Code 62Cx Dept. of Electrical and Computer Engineering Naval Postgraduate School Monterey, California 93943-5000	1
5. Kim, Yong Hong 131-01 Sang-bong Dong 222-40, Dong dae Moon-ku, Seoul, Republic of Korea	10
6. Air Force Central Library Sindaebang Dong, Kwanak Gu, Seoul, Republic of Korea	2
7. Library of Air Force Academy Chongwon Gun, Chung Cheong Bug Do, Republic of Korea	2
8. Lee, Whan S. SMC #1415 Naval Postgraduate Monterey, California 93943	1
9. Lee, Dong I. SMC #1974 Naval Postgraduate Monterey, California 93943	1
10. Lee, Sang S. SMC #1795 Naval Postgraduate Monterey, California 93943	1

1835/7

p

DUDLEY KNOX LIBRARY
NAVAL POSTGRADUATE SCHOOL
MONTEREY, CALIFORNIA 93943-6002

Thesis
K4195 Kim
c.1 A parallel structure
for on line identifica-
tion and adaptive control.

Thesis
K4195 Kim
c.1 A parallel structure
for on line identifica-
tion and adaptive control.

thesK4195

A parallel structure for on line identif



3 2768 000 72887 7

DUDLEY KNOX LIBRARY